

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**DESIGN AND IMPLEMENTATION
OF A PLATFORM INDEPENDENT
PROTOTYPE SPECIFICATION EDITOR**

by

Shen-Yi Tao

September 2000

Thesis Advisor:
Second Reader:

Man-Tak Shing
Thomas Wu

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 4

20001026 080

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2000		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE DESIGN AND IMPLEMENTATION OF A PLATFORM INDEPENDENT PROTOTYPE SPECIFICATION EDITOR			5. FUNDING NUMBERS	
6. AUTHOR(S) Shen-Yi Tao				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) <p>The Computer Aided Prototyping System (CAPS) developed at the Computer Science Department, Naval Postgraduate School is an integrated set of tools that is used for rapid prototyping of real time systems. The PSDL editor, a key component of CAPS, allows user to specify prototype design graphically through data flow diagrams and data flow component property menus, and automatically translates the graphical objects into textual specification written in the Prototype System Description Language (PSDL).</p> <p>This thesis builds upon the previous work done on the CAPS editor design and develops an improved Java based graphic/text editor for the PSDL. New functionality is added to increase the user friendliness of the editor and maintain design consistency in real time. The new enhanced editor provides undo/redo and other essential editing functionality, automatic completion of stream types, as well as automatic checking and propagation of the timing constraints.</p> <p>The new editor is more powerful than ever. It tested successfully in classroom to generate prototype and has been used as a tool for software engineering graduate students to design their computer aided prototype project.</p>				
14. SUBJECT TERMS Computer-aided Rapid Prototyping, Software Specification, Real time system, Graphic Editor, Java			15. NUMBER OF PAGES 302	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE IS INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**DESIGN AND IMPLEMENTATION
OF A PLATFORM INDEPENDENT
PROTOTYPE SPECIFICATION EDITOR**

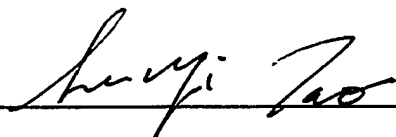
Shen-Yi Tao
Captain, R.O.C. Air Force
B.S., R.O.C Air Force Academy, 1991

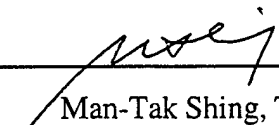
Submitted in partial fulfillment of the
requirements for the degree of

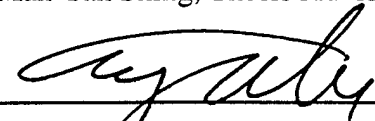
MASTER OF SCIENCE IN COMPUTER SCIENCE

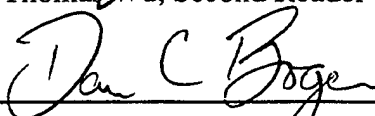
from the

**NAVAL POSTGRADUATE SCHOOL
September 2000**

Author: 
Shen-Yi Tao

Approved by: 
Man-Tak Shing, Thesis Advisor


Thomas Wu, Second Reader


Dan Boger, Chairman
Department of Computer Science

THIS PAGE IS INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	GENERAL INTRODUCTION.....	1
A.	SOFTWARE DEVELOPMENT.....	1
B.	CAPS-A SOFTWARE EVOLUTION TOOL.....	6
C.	RESEARCH GOAL.....	9
II.	EVALUATION OF THE EXISTING EDITOR.....	11
A.	TESTING	11
III.	REDESIGN OF AN ENHANCED EDITOR.....	19
A.	REDESIGN AND USABILITY REQUIREMENTS.....	19
B.	THE INPUT/OUTPUT PACKAGE.....	19
C.	FUNCTIONALITY DESIGN.....	22
D.	ENHANCEMENTS TO THE EXISTING EDITOR.....	28
E.	SUPORT FOR USABILITY.....	36
IV.	IMPLEMENTATION.....	41
A.	STRUCTURE VIEWS.....	43
B.	PACKAGES.....	44
V.	SUMMARY AND CONCLUSION.....	45
A.	USER INTERFACE EVOLUTION.....	45
B.	FUNCTIONAL EVALUATION	46
C.	CONCLUTION.....	46
	APPENDIX A PSDL GRAMMAR.....	49
	APPENDIX B ADA TEMPLATE GRAMMAR.....	55
	APPENDIX C PSDLBUILDERGRAMMAR.JJ AND PSDLBUILDER.JJ.....	57
	APPENDIX D SOURCE CODE.....	103
	LIST OF REFERENCES.....	289
	INITIAL DISTRIBUTION LIST.....	291

I. GENERAL INTRODUCTION

A. SOFTWARE DEVELOPMENT

Computer technology has greatly improved in the last decade. The Windows OS occupies more than eighty percent of the market, and more and more people use computers to work, communicate and play. In contrast, as the PC becomes more inexpensive, the cost of software becomes higher than ever since the user requirements become more complex.

Software development is traditionally thought of as programming. It is very dangerous to think in this way, because programming techniques alone cannot produce complex software that meet customer's need and easily evaluated. To prevent this, a good practitioner should know how to correctly use developing technologies and methods to make his job easier and more efficient.

Be familiar with the process and methods to build the software. Build a team and assign a job to each team member. Define the software requirement and software design specification. Use diagrams to help make things clearer. Ascertain how to write work log and software documents. Facing the more complex software desired by users, a software manager should understand necessary requirements.

1. Team Work

This is a cooperative society. Teamwork is important and absolutely necessary in most cases, especially to a large software system development.

The numbers of employees needed in software development depend upon the number of independent groups of objects. The organization of the employees in each group may be organized like a surgical team. There is a leader who is responsible for the main body of software development, and the other team members assist the leader. Each of team members has a different task.
[Mythical Man-Month]

2. Documentation

A reference manual is necessary for any successful software. The style must be precise, complete, and detailed. A writer should keep its content precise and accurate. The other thing is the use of formal definition to describe the implementation of software or hardware system. Formal definition facilitates the reuse of software implementation. Without formal definitions, it would require re-engineering to recover the system's design before adapting the implementation.

Keeping meeting records and telephones logs for questions and answers are the other important documentation in software development. A formal project workbook should be established at the beginning of a

project. It should include objectives, external specifications, interface specifications, technical standards, internal specifications, and administrative memoranda. Since it is a life-long document, it is important to keep the workbook structure up-to-date and available.

Different levels of documentation are required for software. Different documents depend upon who needs it and what it will be used for. For example, a document for a user should be easy to understand yet interesting. For a manager, document must be concise and detailed. A document can also use graphics to make it as clear as possible. A concept model such as an object model is useful for expressing the concepts among developers. A consistent notation for the software codes can also make program much easier to read and to reuse.

3. Requirements

The hardest single part of building a software system is deciding precisely what to build. In most cases, a client does not understand software programming. Precise communication and clarification of the client's requirements is necessary at the very beginning of the project.

The use appropriate visual aids, such as a data flow diagram, concept models, and class diagram can have a spectacular effect on software development.

4. Architecture and Implementation

One way to make software development efficient is to separate the development process into two phases: architecture design and implementation.

Self-discipline is required when designing software. A software architect who completes his first system design successfully should be careful of his second system design, because his first design experience may be not enough for his second over-ambitions design. A great architect should be conscientious of the particular hazards of the designing system. Paying respect and concentrate on other senior designers' opinions is the other good discipline. Continuing education is necessary to the software developer.

5. Milestone

A milestone of software development sometimes is ambiguous. Because the customer may change requirements in the midst of the development, causing the estimated schedule to slip. A risk assessment plan is necessary since a day-by-day slippage in the schedule will be expensive, and may ultimately cause the project to fail.

6. Modularity

When a large system contains more than 100 thousand lines, tracing codes becomes difficult. Using modular design make building complex software easier and more adaptable to change. Decomposing complex work into its

sub-works results in small modules that are much easier to see than doing the whole thing at a time. All successful software needs change to prolong its usefulness.

7. Modeling

Software is invisible. Using graphs to present software flow of control, flow of data, time sequence, and patterns of dependency is a good way to understand software. One way to make software more visible (understandable) is to select a computer aided programming tool and a high level language which is concise and powerful. Every project is composed of small parts. The best way to complete the whole project is to start from the top, working down to the extended parts. Every designing process must be thought through carefully. Every part should be fully tested separately before the whole system is tested as a whole and every change made in the testing phase shall be recorded. This will save time in the long run.

8. Software Engineering Method and Tools

In modern computer environment, many concepts and technologies, such as object-oriented programming, automatic programming, graphical programming, artificial intelligence, and expert system, etc. are useful. Numerous software concepts and methods are known for increasing efficiency, reducing software development

time, saving money, making valuable development and helping to make a correct evaluation.

A good designer knows how to use effectively software assistant tools to save time and increase productivity.

Good quality software not only reduces testing time, but also maintenance; therefore, a good quality software design can increase productivity. Object-oriented programming can also increase productivity because the module design is easy to reuse, and a clean interface can reduce development complexity. To make our code reusable, a good documentation, as well as through testing is necessary.

B. CAPS - A SOFTWARE EVALUATION TOOL

1. General Description

The Computer Aided Prototyping System (CAPS) is a tool developed by the Software Engineering Group of the Naval Postgraduate School to provide computer-aid in an evolutionary prototyping process. The CAPS allows user to develop prototypes using a Prototype System Description Language (PSDL) and generates control code automatically from the PSDL specification.

The main purpose of a computer-aided prototyping system is to: aid the designer in constructing and evaluating a prototype, provide an execution, debug and monitor capability and support the transformation of a prototype into the corresponding production version of the system.

2. Prototype

Prototypes provide inexpensive demonstration of the essential aspects of the proposed new system behavior. Prototypes are constructed prior to the production version to gain information that guides analysis and design, and supports generation of the production version.

The prototype description language (PSDL), a key foundation of CAPS, enables CAPS the ability to capture the requirements of complex real-time systems, provides uniform conceptual framework and provides high-level

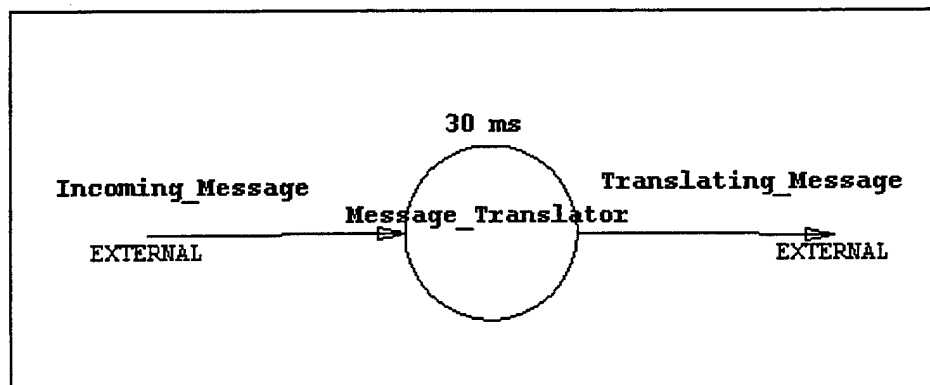


Figure 1 Sample PSDL Graph

system description. Basically, there are two kinds of PSDL elements: operators or types.

PSDL models an operator as an augmented graph - -

$G = (V , E , T(v) , C(v))$

- V = set of vertices
- E = set of edges
- T(V) = set of timing constraints
- C(V) = set of control constraints

A graphic PSDL editor can translate the graphic data flow structure (Figure 1) into a textual description, or vice versa. The PSDL language grammar, as defined in Appendix A, provides the formal basis for the entire translating mechanism.

Each PSDL element contains two parts: specification and implementation.

Specification

- Defines interfaces
- Formal and informal descriptions
- Requirement traces

Implementation

- Architecture description
- Defines decomposition of composite elements
- Code interface descriptions
- Defines atomic elements

A graphic PSDL editor can combine specification and implementation into the graphic structure through a set of pop-up menu.

C. RESEARCH GOAL

Currently CAPS is running on the Unix system only. A portable system is needed for future use in a heterogeneous environment. The Heterogeneous System Integrator (HSI) is an extension of CAPS and HSI will be designed for using on different platforms. Initial development of a portable editor was conducted by Illker Duranlioglu, who built a JAVA implementation of editor. However, the initial development is full of errors. Many major editing functionalities are missing, the translation between the PSDL textual description and the graphic structure is not yet completed, and the user interface need to be made more user friendly.

My principal research objective is to design and develop a robust and user-friendly portable graphic PSDL editor based on the previous work. This thesis builds upon the previous work done on the CAPS editor design and develops an improved Java based graphical/text editor for the PSDL.

The result of this work will allow the editor to automatically generate an error-free PSDL code from the

graphic notations, and automatically generate graphic notations from PSDL code.

The portable PSDL editor will be able to edit both PSDL graphic symbols and PSDL text files. The whole editing process will be checked by a robust PSDL grammar checker to ensure that every translation is correct. The PSDL editor will run on different platforms; and the resultant PSDL files can be imported and reused on different platforms.

New functionality will be added to increase the user friendliness of the editor and maintain design consistency in real time. The new enhanced editor will provide undo/redo and other essential editing functionalities, automatic completion of stream types, as well as automatic checking and propagation of the timing constraints. Data flow components will be created more easily than before, especially for the expert users.

II. EVALUATION OF THE EXISTING EDITOR

A. TESTING

Testing and debugging are required to gain understanding of previous work. Systematically dividing the testing work into categories based on the concept of job/functionality can make redesign easy and clear. Selecting an automatic debugging tool and use the right testing algorithms and methods make things easier and little time consuming. In this testing, certain debugging tools such as Jbuilder 3, Java 2, Java compiler compiler (JavaCC), and Metamata Java analysis tool kit are used. Object-based concepts are also used in this testing. The general outline for this testing and code tracing process are illustrated in Figures 2 and 3.

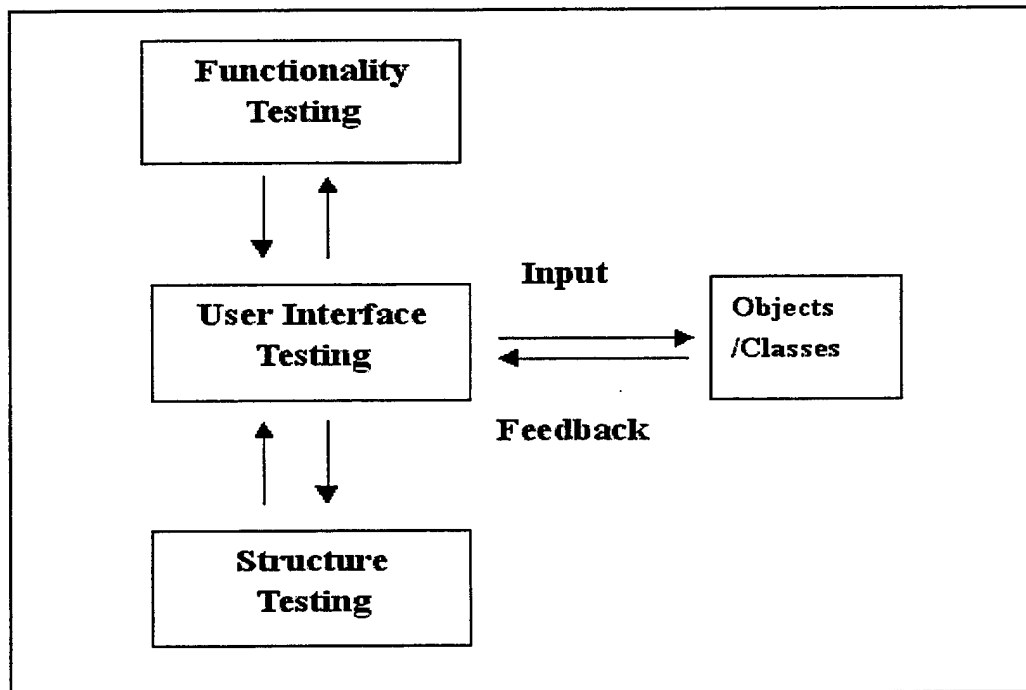


Figure 2 Object Testing

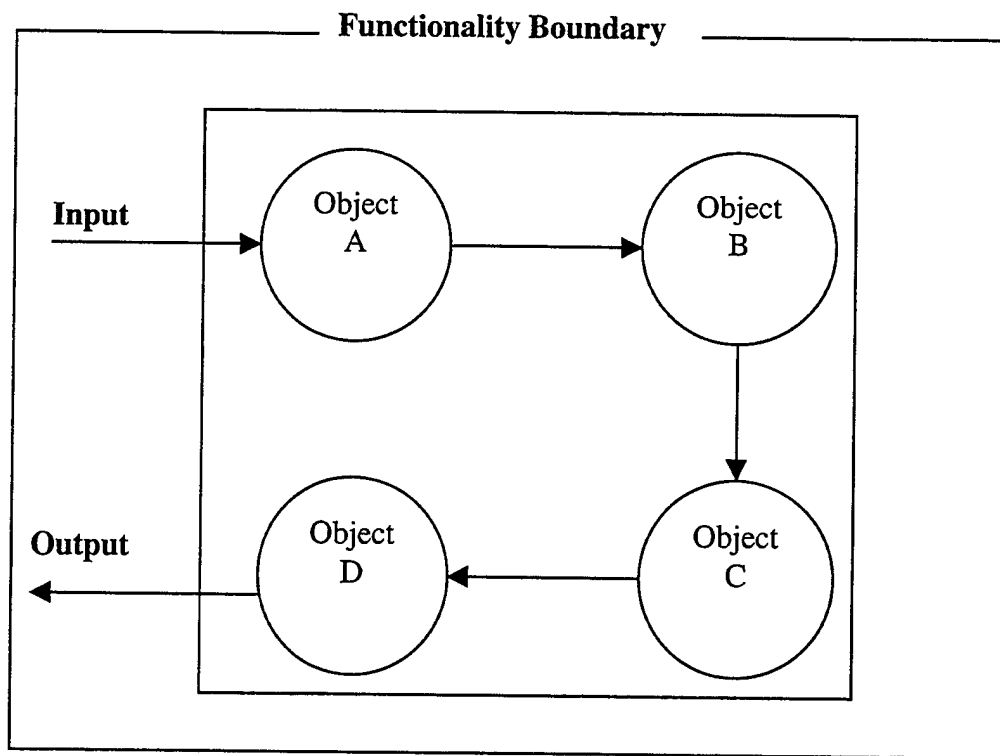


Figure 3 Code Tracing by Functionality

1. Testing Method

The previous program contains approximately five thousand lines of source code and has little documentation. Reengineering is required before designing a new, robust version of the software. The reverse engineering process consists of four steps. The first step is to study the existing Java code structure to gain sufficient background knowledge. Currently, there is no automatic tool to analyze an undocumented function and writing down a complete documentation for it. This may be a good research topic on automatic generation of documentation for an undocumentative function by analyzing its code structure. The second step is to analyze the structure and functionality to ascertain the

original design. The third step, breaking down testing process into small tasks. Every micro-process in one single process or job follows logic sequence or objects' event-triggering sequence. Use of graphic charts make debugging more efficient. The fourth step is to test one job, based on the functionality, at a time. Java VM is able to catch run time exceptions and output a run time error message which is useful for dynamically debugging. The PSDL grammar was rewritten into a JavaCC parser. JavaCC compiles this parser and automatically generated the PSDL package in the Java language. The PSDL package provides a mechanism for checking user input from user interface and provides very useful information for debugging when errors occur between PSDL texts and PSDL graphs.

Finally, document the testing result. In this part, only bugs in the code shall be recorded. The design inconsistency, logic errors and function incomplete become redesign requirement to be solved later. This is for keeping consistency in code testing.

2. Problems with Existing Design

All faults found in this testing have been corrected. Here we list some significant problems found, which can serve as a reference for future evolution of the editor software.

a) Problem 1

Changing the text in Vertex property dialog results in an error message or system exception error when saving the change.

Problem: No checking for user-input restriction, checking methods are wrong or the functionality is incomplete in the PSDL grammar file.

Requirement type: GUI and JavaCC grammar files design.

b) Problem 2

An error message is displayed or the program hangs up when try to save/open current PSDL text files.

Problem: Design is incomplete in the JavaCC grammar files and the code modules related to generating PSDL code.

Requirement type: grammar files and function redesign.

c) Problem 3

The generated PSDL files cannot be used in different platforms.

Problem: The generated PSDL code is incorrect or insufficient.

Requirement type: grammar files and function redesign.

d) Problem 4

A lot of minor errors in user interface operations, such as resizing or changing the configuration of the data flow diagrams in drawing area, e.g. changing the focusing on data flow component (DFC) navigator, drawing panel do not refresh, missing prompt for "save are required", etc.

Problem: GUI design error.

Requirement type: GUI redesign.

e) Problem 5

Drawing a stream from the EXTERNAL source vertex to a vertex may cause the program to hang up in some situation .

Problem: Functionality is incomplete and will cause run time exception.

Requirement type: function redesign.

f) Problem 6

Mouse drag may cause the program to hang up, occasionally in the "select all mode".

Problem: mouse drag event design incomplete and functionality design error.

Requirement type: redesign mouse event and functions.

g) Problem 7

Saving a stream with the EXTERNAL source vertex will cause grammar check error in some situations.

Problem: functionality incomplete in the code generation.

Requirement type: function design.

h) Problem 8

The tree panel and drawing panel does not work together. There are consistent problems when the focus on one or the both is changed.

Problem: Inconsistency in GUI.

Requirement type: GUI redesign.

i) Problem 9

Clicking the vertex on the tree panel that has an edge with the External source vertex can cause tree folder collapsing failure.

Problem: Errors design related to tree panel events.

Requirement type: redesign tree panel events.

j) Problem 10

When selecting a component in the tree panel, the drawing panel does not focus on the selected components.

Problem: GUI design incomplete on tree event.

Requirement type: redesign tree panel events.

k) Problem 11

When selecting a leaf node in the tree structure and then selecting its parent vertex, the previous selecting marker does not disappear.

Problem: Function design incomplete on selection.

Requirement type: Function redesign.

THIS PAGE IS INTENTIONALLY LEFT BLANK

III. REDESIGN OF AN ENHANCED EDITOR

A. DESIGN AND USABILITY REQUIREMENTS

Code testing has found a lot of GUI redesign requirements. Furthermore, new requirements are added to enhance the usability of the editor. The goal is to provide an error free, stable rapid prototyping environment to increase the editing efficiency and productivity of the software engineer.

B. THE INPUT/OUTPUT PACKAGE

1. Graphic Notations

PSDL is a text-based language, which is designed to describe the specification of a real-time system. Data flow diagram was used to represent PSDL code structure. There are two kinds of data flow components: one is the vertex, which includes both operator and terminator, and the other is the edge (Figure 4). Every Data flow component (DFC) has its own PSDL property. Moreover edges with the same name must have the same property, and vertex must satisfy the real time constraints derived from their parent vertices.

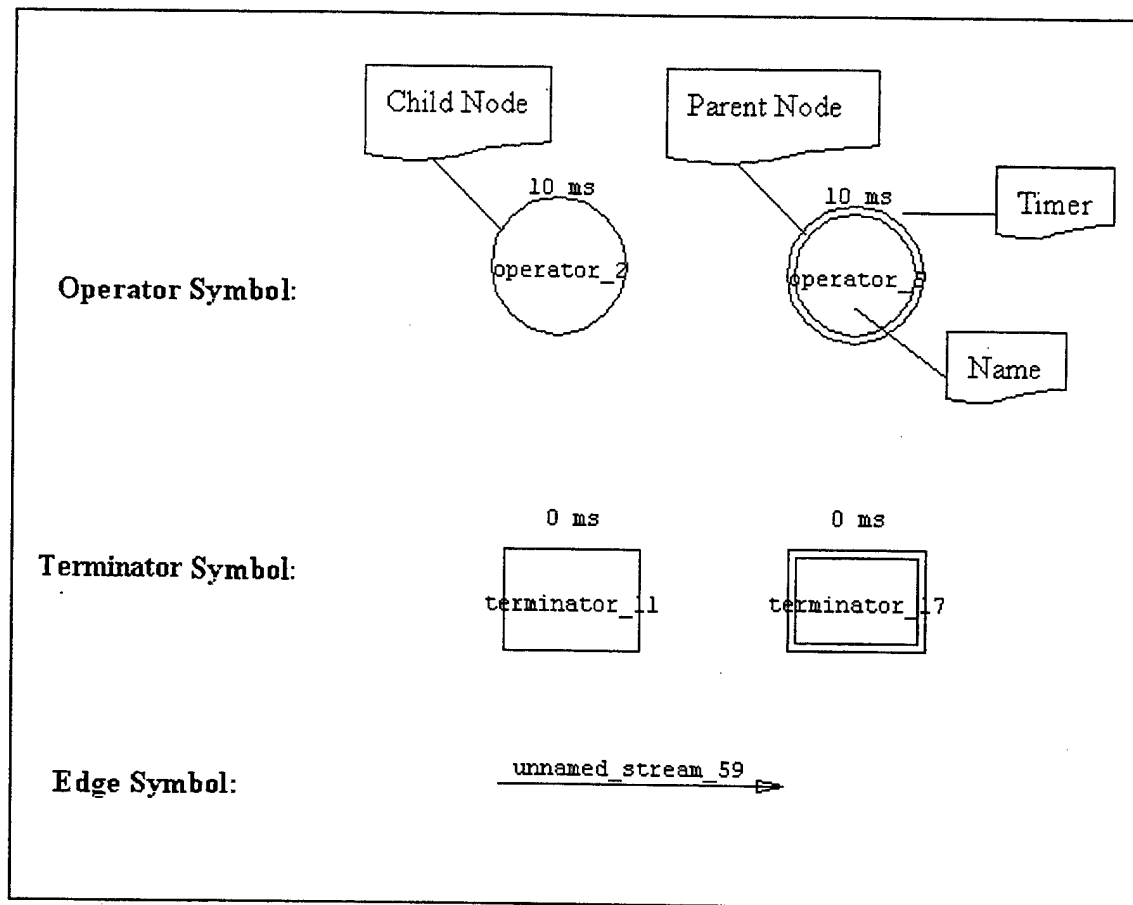


Figure 4 PSDL Graphic Notations

2. PSDL Packages

There are two kinds of PSDL packages. One is for "checking a file", which means to verify a PSDL text file. The other is for "opening a file", which means to translate PSDL text file to PSDL graphic notations.

Before redesigning the PSDL packages, some tasks must be done first. PSDL grammar is the basis of this PSDL editor design. It provides the fundamental concept of how the result will appear after the editor translates graphic symbols to text file and vice versa. Since the PSDL grammar is written in the BNF form, Java compiler

compiler (JavaCC), a Java language based parser generator is selected to translate a grammar to Java code. We first rewrite the PSDL BNF file into a JavaCC parser file, then use the JavaCC to automatically translate this file into the PSDL Packages. For "opening a file", customized Java codes was added to JavaCC parser files to generate graphic notations. The JavaCC reads the tokens sequentially. If the tokens match the PSDL grammar, the parser will generate a package; otherwise, the parser will display compiling error messages to system screen.

Keep PSDL BNF file as a reference. It is much easier to be read than both the JavaCC parser files and the PSDL Packages.

The redesign is to find faults from previous JavaCC files and design the correct and detail version, because the previous files are incomplete and error prone. The result enables translation between text and graphic notations without errors and provides a PSDL grammar monitor during translation. The redesign process is illustrated in Figure 5. Some code for the parser and "text to graphic" translation can be found in PSDLGrammar.jj and PSDLBuilder.jj in Appendix C.

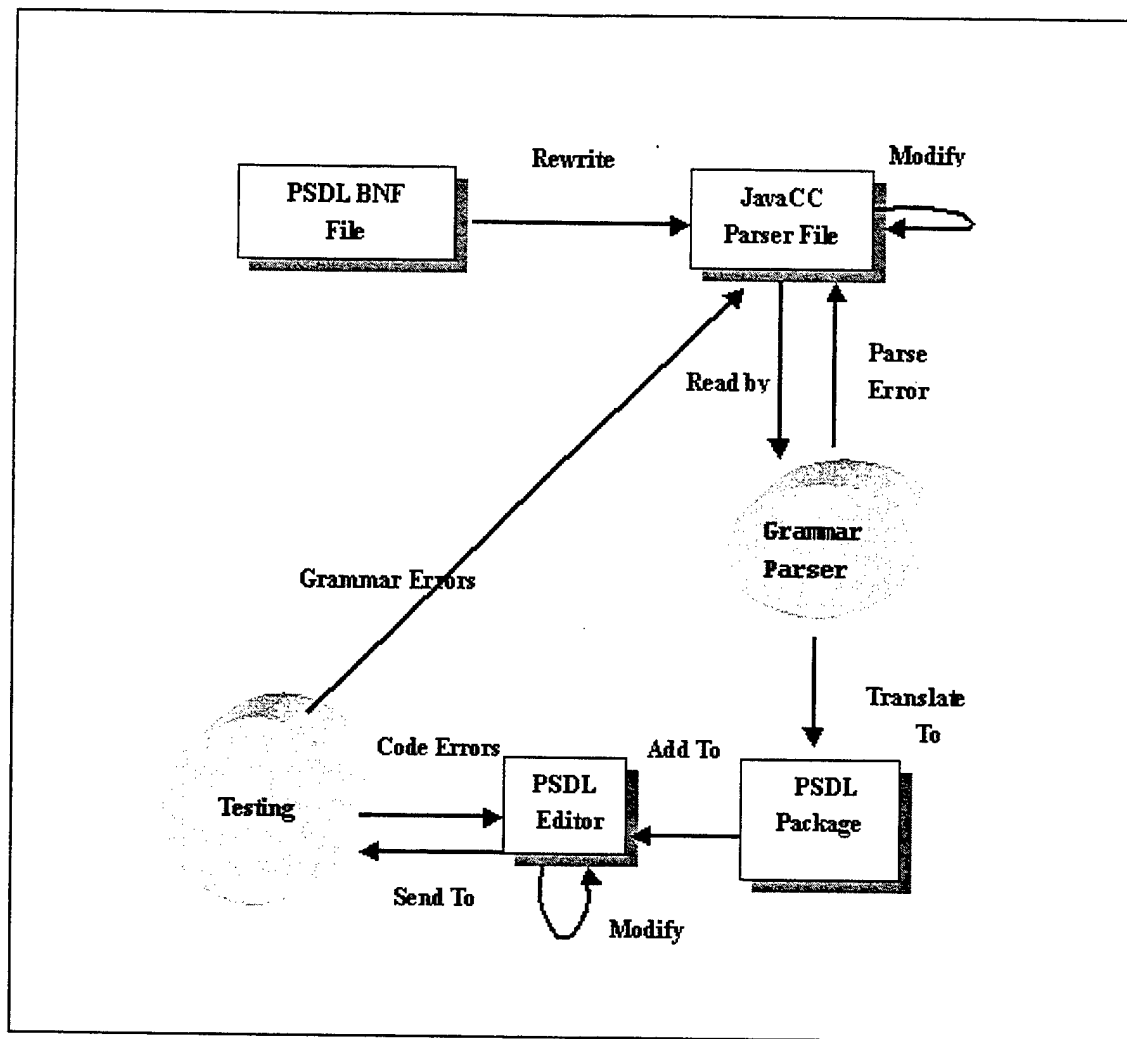


Figure 5 Grammar File Redesign

C. FUNCTIONALITY DESIGN

1. Structure Design

PSDL prototypes are written in formal structure, with graphic structures that look like multi-level data flow diagrams, where every vertex in one level can be decomposed into its sub-level data flow graph. A simple example is shown in Figure 6. Java standard JTree package is used to build this tree structure.

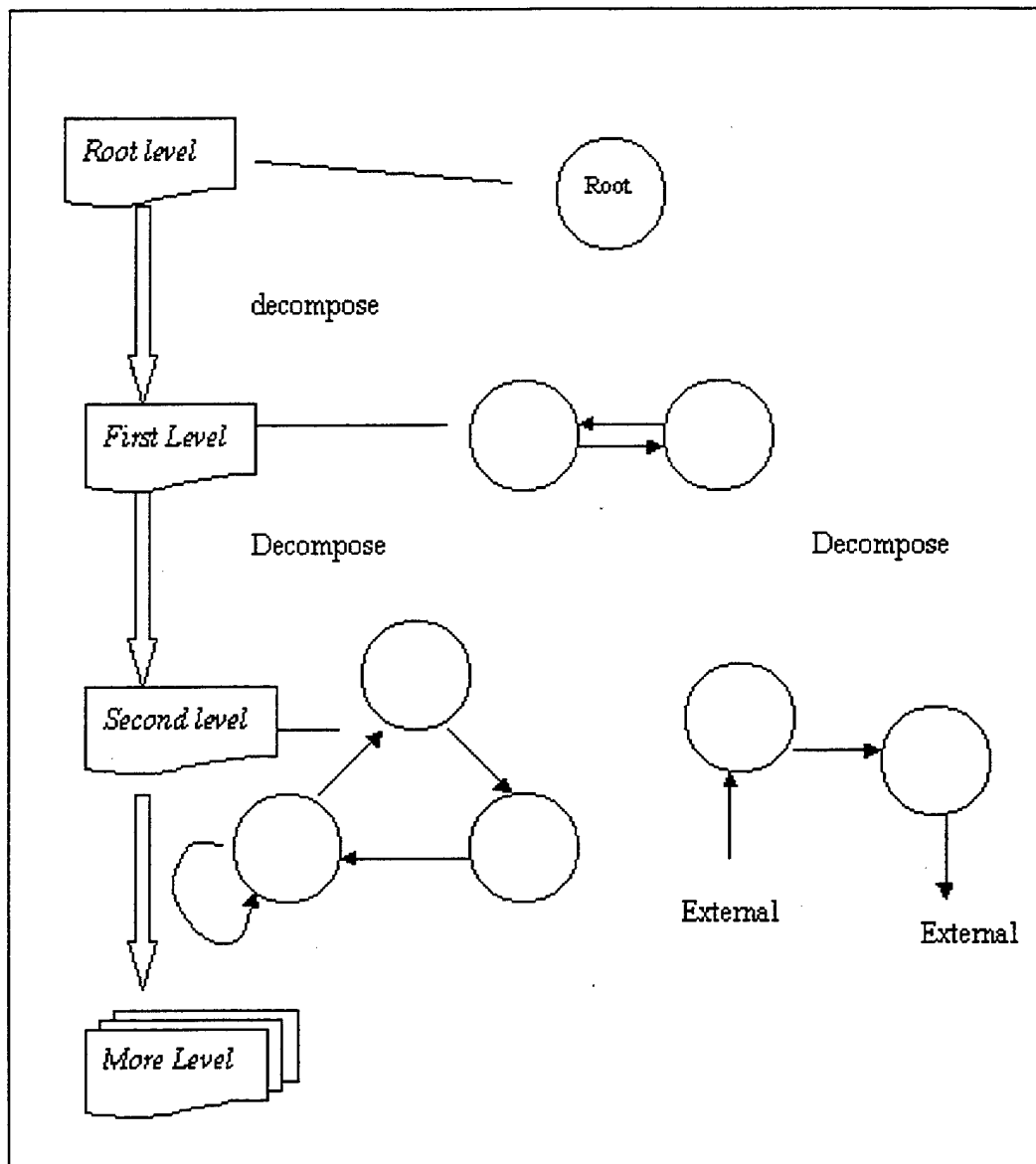


Figure 6 A PSDL Data Flow Components Diagram

2. Automatic Code Generation of PSDL Code from Graphic Notation

This code generation includes five transformation steps. First, read graphic notations and stores them into a temporary vector. Second, "PSDL structure checker" checks the structure of graphic notations in the vector. Third, read graphic notations and translate them to PSDL code. Fourth, "Grammar monitor" examines the PSDL code.

save the PSDL code into a text file. An example of a generating file for Figure 7 is listed as Table 1.

Some code for checking and generating PSDL code from graphic descriptions can be found in CreatePsd1.java, DataFlowComponent.java, Vertex.java and Edge.java in Appendix D.

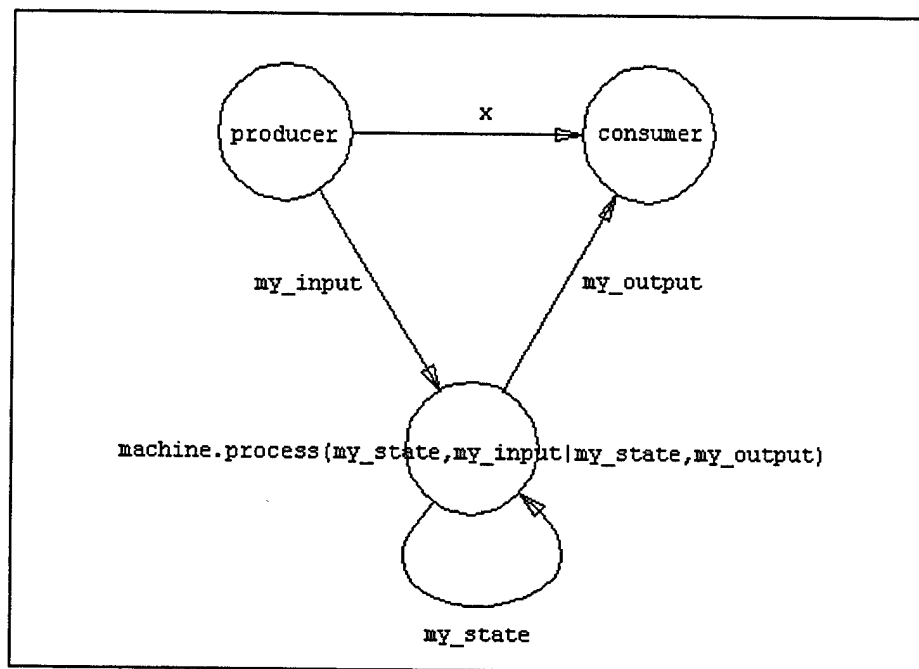


Figure 7 A Sample PSDL Data Flow Diagram

```

OPERATOR sample_2
SPECIFICATION
END
IMPLEMENTATION
GRAPH
  VERTEX producer_25_24
    PROPERTY x = 157
    PROPERTY y = 177
    PROPERTY radius = 35
    PROPERTY color = 62
    PROPERTY label_font = 5
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY met_font = 5
    PROPERTY met_unit = 1
    PROPERTY met_x_offset = 0
    PROPERTY met_y_offset = -40
    PROPERTY is_terminator = false
    PROPERTY network_mapping = ""

```

Continue next page

```

VERTEX consumer_28_27
  PROPERTY x = 355
  PROPERTY y = 175
  PROPERTY radius = 35
  PROPERTY color = 52
  PROPERTY label_font = 5
  PROPERTY label_x_offset = 0
  PROPERTY label_y_offset = 3
  PROPERTY met_font = 5
  PROPERTY met_unit = 1
  PROPERTY met_x_offset = 0
  PROPERTY met_y_offset = -40
  PROPERTY is_terminator = false
  PROPERTY network_mapping = ""
VERTEX machine.process_31(my_state,
my_input|my_state, my_output)
  PROPERTY x = 254
  PROPERTY y = 344
  PROPERTY radius = 35
  PROPERTY color = 52
  PROPERTY label_font = 5
  PROPERTY label_x_offset = 0
  PROPERTY label_y_offset = 0
  PROPERTY met_font = 5
  PROPERTY met_unit = 1
  PROPERTY met_x_offset = 0
  PROPERTY met_y_offset = -40
  PROPERTY is_terminator = false
  PROPERTY network_mapping = ""
EDGE x producer_25_24 -> consumer_28_27
  PROPERTY id = 33
  PROPERTY label_font = 5
  PROPERTY label_x_offset = 0
  PROPERTY label_y_offset = 0
  PROPERTY latency_font = 5
  PROPERTY latency_unit = 1
  PROPERTY latency_x_offset = 0
  PROPERTY latency_y_offset = -40
  PROPERTY spline = ""
EDGE my_input producer_25_24 ->
  machine.process_31(my_state, my_input|my_state,
  my_output)

  PROPERTY id = 35
  PROPERTY label_font = 5
  PROPERTY label_x_offset = -79
  PROPERTY label_y_offset = 5
  PROPERTY latency_font = 5
  PROPERTY latency_unit = 1
  PROPERTY latency_x_offset = 0
  PROPERTY latency_y_offset = -40
  PROPERTY spline = ""
EDGE my_output machine.process_31(my_state,
my_input|my_state, my_output) ->
  consumer_28_27
  PROPERTY id = 37
  PROPERTY label_font = 5

```

Continue next page

```

        PROPERTY label_x_offset = 0
        PROPERTY label_y_offset = 0
        PROPERTY latency_font = 5
        PROPERTY latency_unit = 1
        PROPERTY latency_x_offset = 0
        PROPERTY latency_y_offset = -40
        PROPERTY spline = "π"
    EDGE my_state machine.process_31(my_state,
        my_input|my_state, my_output) ->
        machine.process_31(my_state,
        my_input|my_state, my_output)
    PROPERTY id = 43
    PROPERTY label_font = 5
    PROPERTY label_x_offset = -15
    PROPERTY label_y_offset = 17
    PROPERTY latency_font = 5
    PROPERTY latency_unit = 1
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = -40
    PROPERTY spline = "238 393 245 405 275 413
    299 400 295 383 "
DATA STREAM
    x : integer,
    my_input : integer,
    my_output : integer,
    my_state : integer
CONTROL CONSTRAINTS
    OPERATOR producer_25_24

    OPERATOR consumer_28_27

    OPERATOR machine.process_31
    (my_state,my_input|my_state,my_output)
END
OPERATOR producer_25
    SPECIFICATION
        OUTPUT x : integer
        OUTPUT my_input : integer
    END
IMPLEMENTATION Ada producer
END

    OPERATOR consumer_28
    SPECIFICATION
        INPUT x : integer
        INPUT my_output : integer
    END
IMPLEMENTATION Ada consumer
END

TYPE machine
    SPECIFICATION
        OPERATOR process
        SPECIFICATION
            INPUT my_input : integer
            INPUT my_state : integer
            OUTPUT my_output : integer
            OUTPUT my_state : integer
        END
    END
IMPLEMENTATION Ada machine
END

```

Table 1 Sample PSDL File

D. ENHANCEMENTS TO THE EXISTING EDITOR

1. Undo and Redo

Undo and redo is a complex usability requirement in the PSDL editor design (Figure 8). The reasons are: (1) PSDL structure is a complex tree; (2) every edge between the two tree nodes/vertices is also a part of the PSDL definition and edges are directed; (3) modifying any vertex also affects its related edges and their related vertices; and (4) modifying any edge also affects its related vertices.

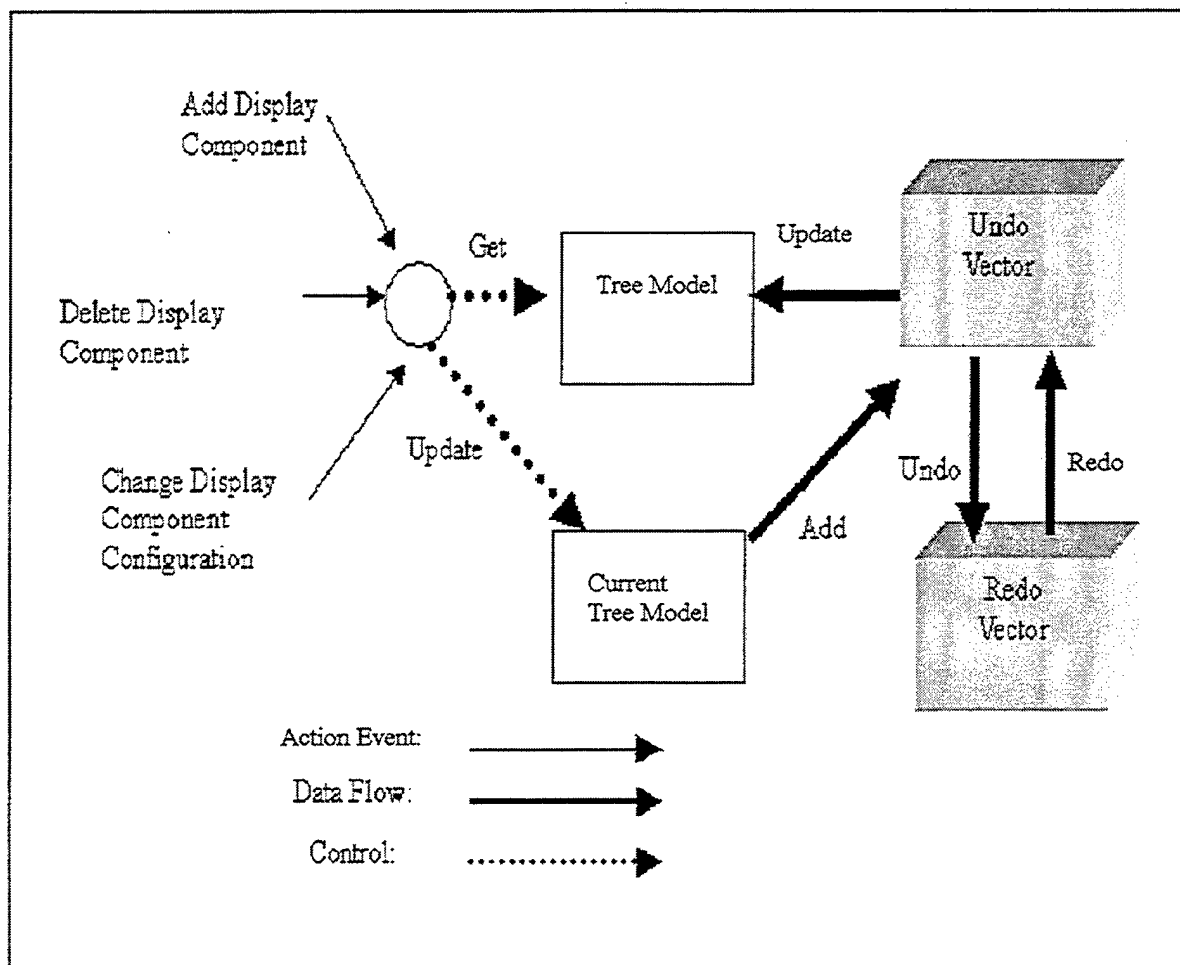


Figure 8 Undo/Redo Processes

a) User view

From the user view, at least three requirements are needed in the undo and redo functionality:

- Save the current level display components in draw panel into a vector.
- Redraw/update the current draw panel and tree field.
- Search and retrieve old level components from the vector and update current screen.

b) Structure view

From the data flow components (DFC) structure view, using java.swing.tree package, a Default Mutable TreeNode component provides a good way for PSDL editor tree structure implementation. In this presentation, every data flow component is abstractly represented as a tree node. A data flow component has exactly one parent DFC and may have zero or more children DFC. Since DFC is extended from DefaultMutableTreeNode, it inherits a lot of useful operations for modifying and checking status of its parent DFC and children DFC. It also inherits operations for examining the whole tree that the DFC is a part of. The DFC tree is the location in which all DFCs can be reached by decomposing from the root DFC. In this presentation, there is only one root node in a PSDL program. All tree nodes (DFC) share the same root node (DFC).

The most unique idea here is that every edge itself (between two nodes) is also considered as a DFC. Every edge has its direction. So every edge abstractly contains its own source vertex and destination vertex. Consequently every vertex may have zero or more in-edges and out-edges. This greatly increases the difficulty in traveling between tree nodes, especially for creating the undo and redo functionality.

The "=", "Clone ()" and "equal ()" function in Java sometimes have insufficient functionality. The problem is the verification of two data flow with the same property. This will happen when creating a duplicated node or tree to support the undo/redo operation. We found out that it is better to use unique ID stamp instead of "equal ()" function to find the target data flow component. The "clone ()" function is insufficient for a duplication complex tree structure and may cause a lot of unpredictable problems such as two object sharing same memory address when cloning an target object which has static value, array, or vector data member which use a fixed address/pointer. In this presentation, a lot of patches were created to patch these problems.

c) Time view

From the user view, the timing to undo or redo can be described in the time of change of graphic

notations on drawing panel. Since the change of graphic notations is visible, it is easy to understand.

Time to undo:

- After adding new graphic notations.
- After deleting exiting graphic notations.
- After changing the location of graphic notation.
- After the "MET label" of graphic notation changed location.
- After the "name label" of graphic notation changed location.
- After resizing vertex.
- After reshaping edge.
- After redoing one or all graphic notations.

Time to redo:

- After undoing one or all-graphic notations.

2. Edge property consistency

In the PSDL prototype, edges are allowed to have the same name in the same level or in different levels of the tree structure. All the edges with the same name are regarded as the same data stream and have the same property. So, the edge inconsistency problem may occur when one of these edges changes its property. In this design, edges consistency is automatically maintained. The user does not need to worry about this problem. This design includes two cases: (1) dynamically detects the

edge's name changed and automatically fill in the edge property if it has the same name with an existing edge; and (2) when any edge's property changed and the OK button was clicked, searching all the edges existed to find every edge with the same name but different properties. If such edge is found, pop up an edge property table (Figure 9) to allow user to select a correct property.

Stream Type	State Stream?	Initial Value	Latent Value	Latent Unit
boolean	YES	1		
int	NO			

OK

Figure 9 Edge Property Check Table

3. Real Time constraints

According to PSDL, there are three types of timing constraints: Non-time-critical, Periodic and Sporadic, and three type of triggering conditions: unprotected, trigger by all and trigger by some.

The editor must be designed for user to easily specify real time constraints. It should also provide local timing constraint check.

An embedded real time software system must satisfy a set of real time constraints related to property and

performance, so the local and global timing check is required. In this project, we include the following local timing checks; where MET stands for maximum execution time, FW stands for finishing within, PER stands for period and MCP stands for minimum calling period:

- For all periodic operators

(1) $MET \leq FW$

(2) $MET \leq PER$

- For all sporadic operators

(1) $MET \leq MRT$

(2) $MET \leq MCP$

For example, invoking the "check timing" button on the operator with the following constraints (Table 2) will result in the report shown in the Figure 10.

<i>OPERATOR ROOT</i>	<i>OPERATOR producer</i>
<i>⋮</i>	<i>SPECIFICATION</i>
<i>CONTROL CONSTRAINTS</i>	<i>MAXIMUM EXECUTION TIME 1 ms</i>
<i>OPERATOR producer</i>	<i>END</i>
<i>PERIOD 3 ms</i>	
<i>FINISH WITHIN 2 ms</i>	
<i>END</i>	

Table 2 A Sample Real Time Constrains

TIMING CHECK TABLE for test						
Vertex Name	Timing Type	1.MET <= FW	2.MET <= PER	3.MET <= MRT	4.MET <= MCP	Pass/Fail
producer	PERIODIC	Y	Y			P

Figure 10 Edge Property Check Table

Figure 10 shows that producer has periodic timing and it satisfies the local periodic timing rule (1) MET <= Finish Within and (2) MET <= Period, so it passes the check.

4. Graphic user interface design

a) Tool Bar

1	Draw an operator into the Drawing Panel	7	Open id-list editor to view or edit Timers
2	Draw an terminator into the Drawing Panel	8	Open text editor to view or edit Informal Graph Description
3	Draw an stream into the Drawing Panel	9	Undo an editing step
4	Select a component from the Drawing Panel	10	Redo an editing step
5	Open text editor to view or edit Data Types	11	Check local timing
6	Open text editor to view or edit Parent Specs		

Figure 11 Tool Bar

The main tool bar includes five menu items and eleven buttons (Figure 11). A typical prototype design only needs to use the tool bar items to achieve high productivity. Adding a data flow component only needs to "click and paste". Undoing and redoing an editing step needs only one mouse click. Verifying local timing step needs one mouse click to display a verified information. Inspecting data type and parent specification needs one mouse click to display a list, and setting up times need one mouse click to display a setting dialog. The button names are shown in Figure 11.

b) Tree node navigator

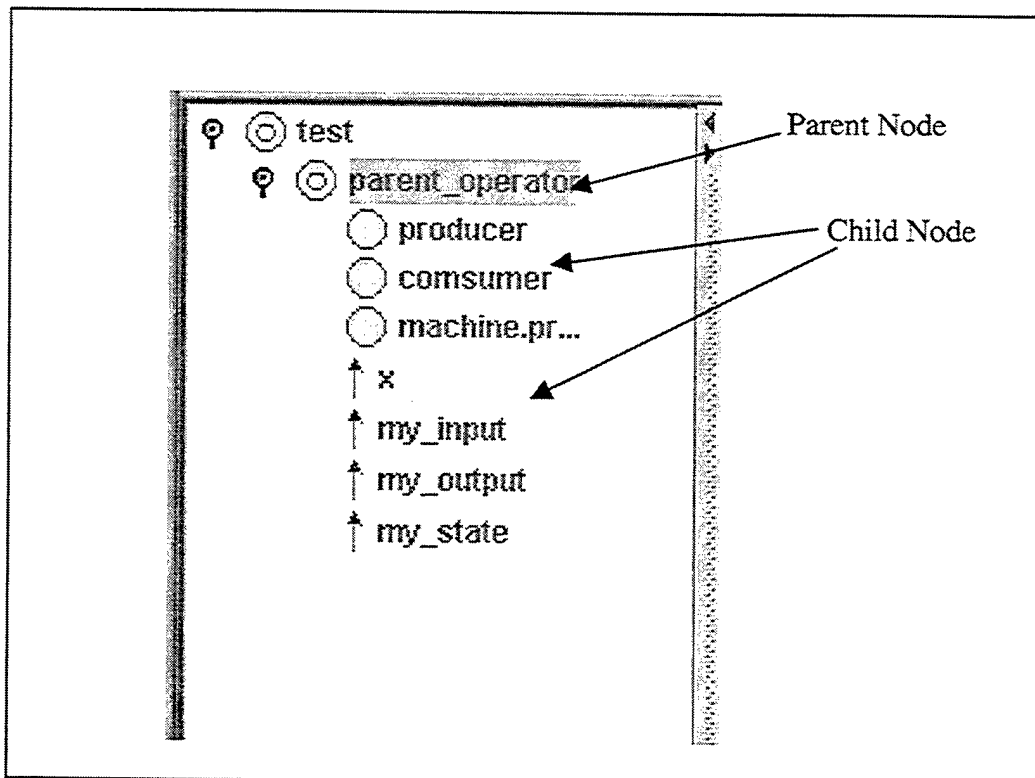


Figure 12 Tree Node Navigator

The tree panel is designed with JAVA JTree. Derived from a general concept of the typical file browser, the folder represents a parent DFC and the file represents child DFC (Figure 12).

The major difference here is that the parent node is a vertex itself and is also a child vertex of its parent vertex. A child DFC can be a vertex or edge, and can be decomposed to its sub-level DFCs if it is a vertex. To decompose a vertex, simply click on the vertex node.

E. SUPORT FOR USABILITY

a) *Automatic Error Checking*

PSDL Grammar monitor provides an excellent debugging support during PSDL editor testing phrase. Before saving PSDL codes to file, the PSDL codes are verified for correct syntax. And if the code is not correct, the PSDL file will not be saved and a grammar error information will be displayed. A sample error message is shown in Figure 13. This enhances the quality of the prototype produced.

When saving the PSDL file, the PSDL data flow diagram will be checked to ensure that the data flow diagram has the correct structure. If the structure is not correct, a structure error message will be displayed (Figure 14).

Furthermore, the editor also performs instant checks during every user input to prevent human error, and an error message will be displayed if error is detected. A sample error message is shown in Figure 15.

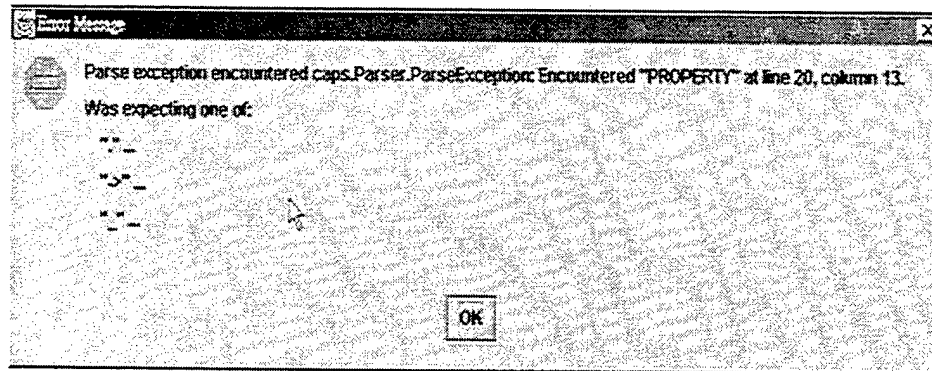


Figure 13 Grammar Error Message

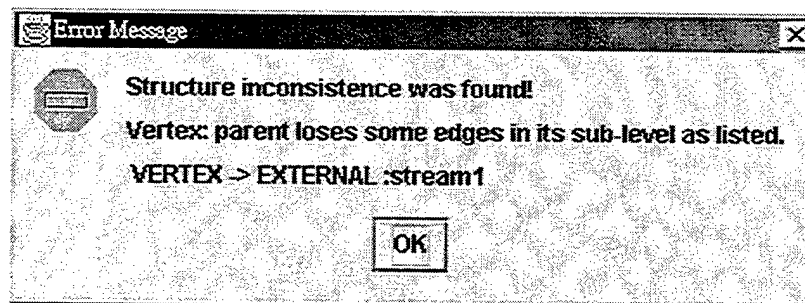


Figure 14 Structure Error Message

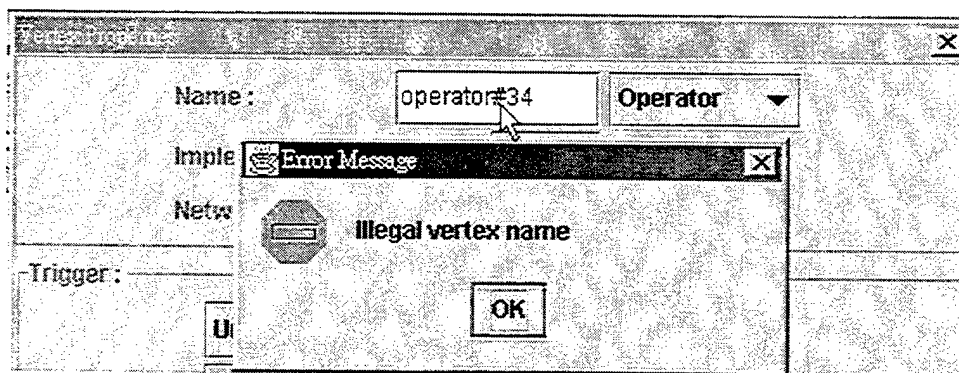


Figure 15 Input Check Error Message

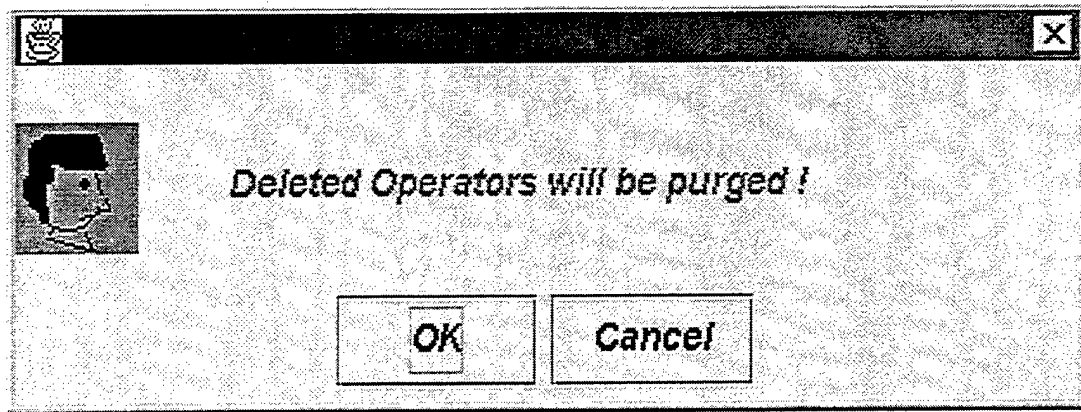


Figure 16 Input Check Error Message

b) Warning Message

Warning message is displayed if the user action will cause permanent change in the design and reset the undo vector. A sample-warning message is shown in Figure 16.

c) Edge trace

Edge Drawing is by collecting points of mouse clicks and then trimming those points into a smooth shape (shown as Figure 17). User can press the Esc key to cancel an unfinished edge drawing. Furthermore, any edge that does not reach a reasonable end point will be detected and erased when cursor left the drawing panel. This helps the system to clean up debris and prevent the run time exceptions.

d) Decompose

Click on the vertex tree node on the tree navigator will decompose this vertex into its sub-level data flow components. This reduces user effort to

navigate the multi-level design and improves productivity. The other way to decompose is by selecting the "Decompose" menu item in the "PSDL" menu on the editor.

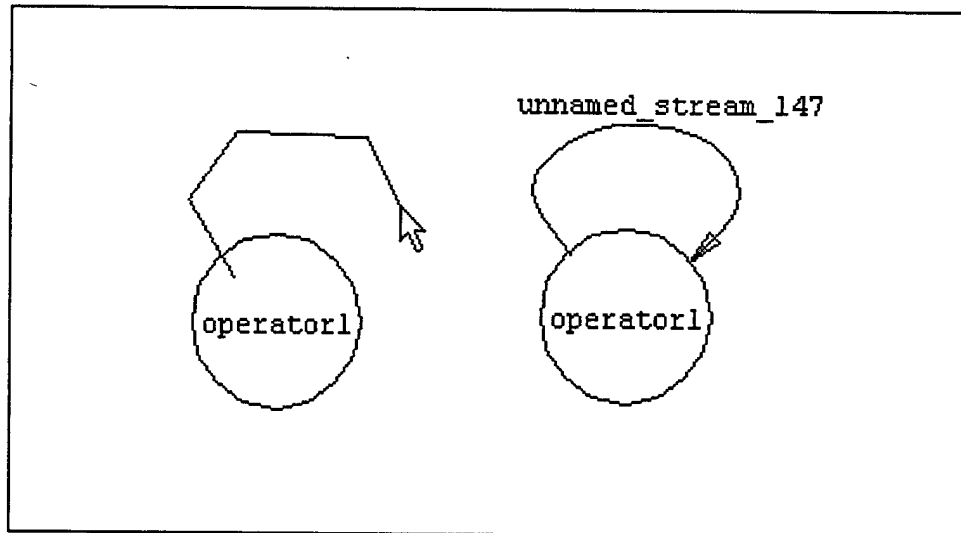


Figure 17 Drawing Edge

e) Adding data flow components

To add a desired Vertex component, simply clicks on the "OPERATOR" labeled button on the tool bar and then click on the correct drawing area. Each click will generate one component. To add an edge, click the "stream" button on the tool bar, then click on the desire position on the drawing area. Edges are generated via click-points-accumulation. A right mouse click on the open drawing panel generates an External sink vertex. An edge can be drawn between two vertices, between vertex and external, or as a loop to and from the same vertex. To cancel the current selected data

flow component, click the "select" button on the tool bar.

f) *Select, Move ,Resize, Delete Components*

To select a data flow component on drawing panel, simple click on it. To select all components, choose the "select all" menu item in the edit menu. To move one or all selected components, simply drag them to the correct place. To resize a selected component, drag one of its handles to change the size or shape of selected component. To delete selected components, hits the "Del" key or choose the "delete" menu item in the "Edit" menu.

g) *Setup Vertex Property*

Properties of the data flow component are enforced through a pop-up dialog, and all user inputs will be guarded and checked. Guarding means preventing user from typing wrong information. Checking means verifying the vertex property syntax when the user hits the "OK" button of the dialog boxes. The source code of the property dialogs is in the VertexProperty.java file. Instead of reading to read the code documentation, user should refer to the PSDL grammar file to understand the input restriction.

Since the PSDL editor is based on the PSDL grammar, Property dialogs are restricted by PSDL grammar, which can be found on the PSDL grammar file (Appendix A).

h) Setup Stream Property

Stream is the other name of edge. Basically, design of the stream property setup is similar to the vertex property setup; but the items of the stream property are much fewer than those of vertex property. The source code of the stream property setup is in the EdgeProperty.java file.

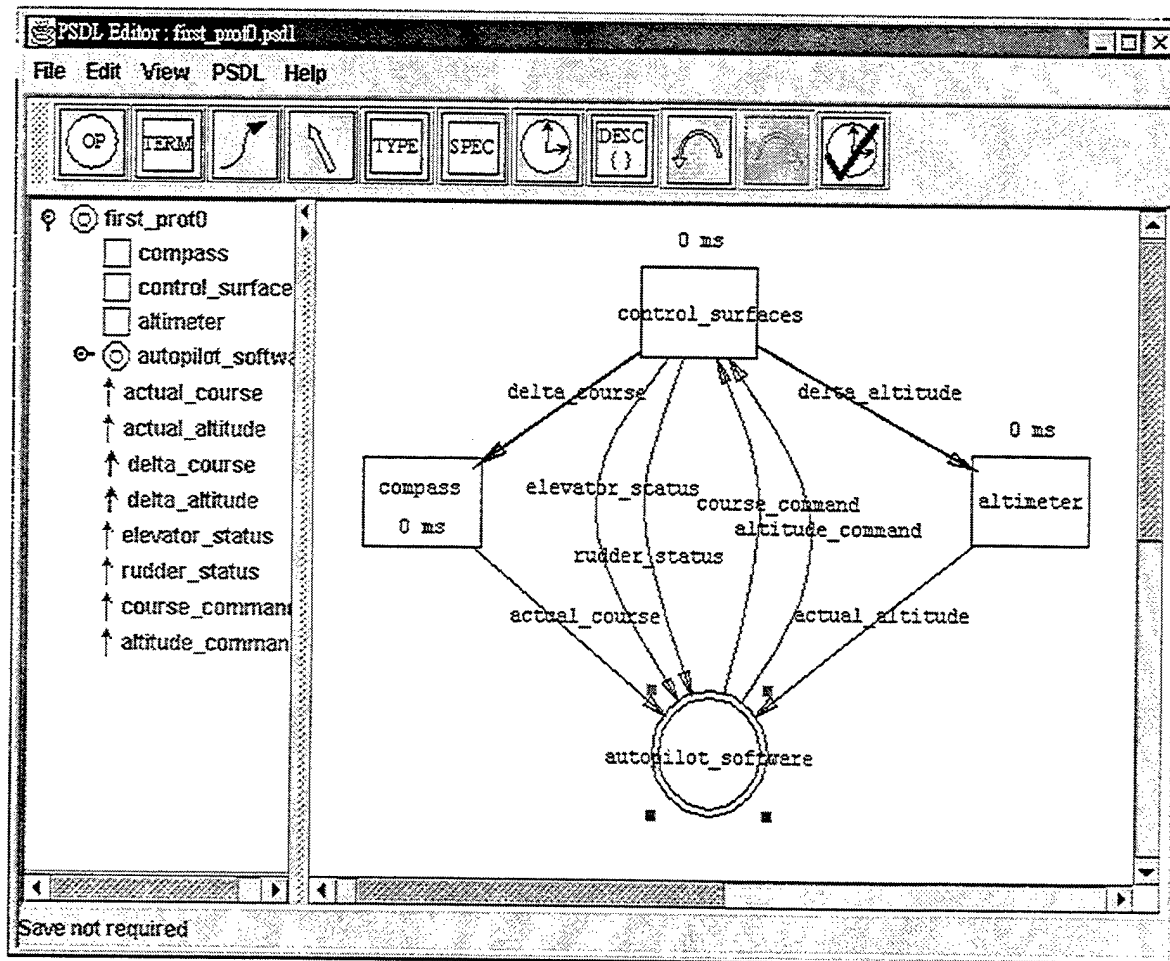


Figure 18 The Portable PSDL Editor

IV. IMPLEMENTATION

A. STRUCTURE VIEW

For developing consistency, the program structure of this release is similar to the previous one. Most major objects were debugged, modified or enhanced, and some independent objects were added. The program structure is illustrated in the Figure 19.

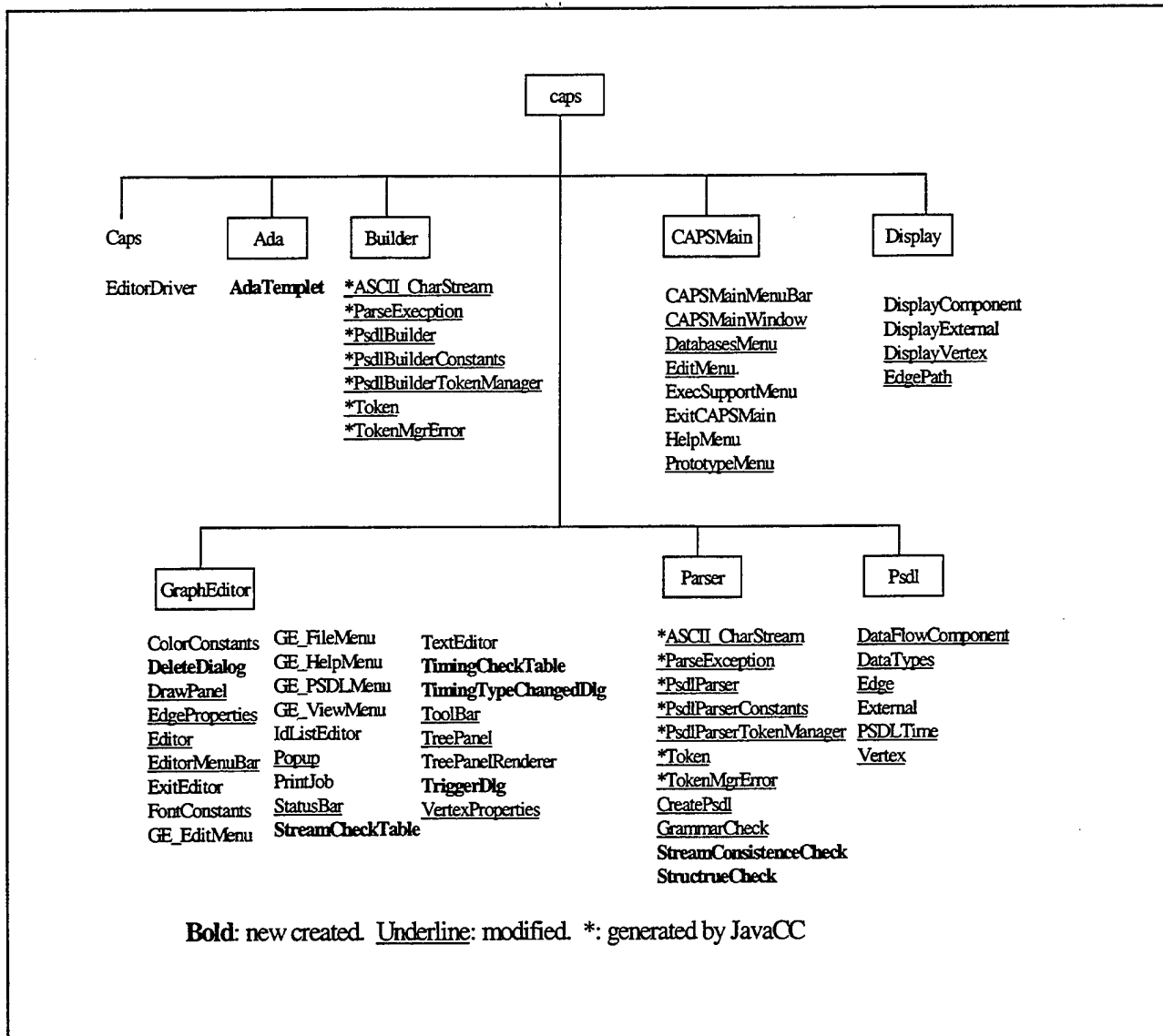


Figure 19 PSDL editor program structure

B. PACKAGES

1. Package caps.Ada

This package contains the classes for generating Ada templates. CAPS will use the Ada template file with the PSDL file to generate the Ada code. This package is a new addition created to allow CAPS to generate Ada code for the PSDL components with an Ada implementation in the prototype. The Ada template grammar can be seen in the Appendix B.

2. Package caps.Builder

The Builder package contains the objects that are responsible for reading a PSDL file and to construct the data structures that are used to keep track of the graphic notations in the PSDL editor. Changes made in this package include: (1) generating data flow diagram for complex PSDL structure, (2) generating graphic notation properties for complex PSDL property definition, (3) generating PSDL real time properties, and (4) testing and debugging the previous PSDLBuilder.jj file.

JavaCC generates all classes from the PSDLParser.jj file for this package. The parser file is in the PSDLBuilder.jj in Appendix C . The grammar rule that is used to create the parser is presented in Appendix A.

3. Package caps.CAPSMainWindow.

The CAPSMainWindow contains the main program that is necessary to run the PSDL editor. Only minimum changes,

which are related to creation of an Ada template, have been made to this package.

4. Package caps.Display

The Display package contains the classes that are used to model the graphic notation of the PSDL prototype. The instance of this model will be used as an abstract display notation on the screen. This package has been tested and debugged.

5. Package caps.GraphEditor

The GraphEditor package contains the classes that implement the user interface components of the PSDL editor. This design has been modified to: (1) add network label and "hard and soft real-time constraint" radio button to vertex property, (2) enhance user input syntax check, (3) handle tree panel navigator event, (4) add stream consistency propagation and checks, (5) add vertex consistency checks, (6) add undo and redo functionality, (7) allow deletion or move of individual or all graphic notations, (8) enhance vertex and stream property menus, (9) add line tracing while drawing an edge, (10) add local and global real time consistency checks on multi-level components, (11) add data flow diagram structure check, (12) improve the user interface events of the vertex property dialog, (13) add user input warning messages, and (14) test and debug the previous package.

6. Package caps.Parser

The Parser package is used by the PSDL editor to validate the user input and the generated PSDL file. This design has been modified to: (1) add the PSDL grammar check to correctly response on the user input and the PSDL file, (2) add the real time properties check, (3) add stream consistency check, (4) add data flow diagram structure check, and (5) test and debugging the previous PSDLParser.jj file. Some classes are generated by JavaCC and some are not. (Refer to Figure 19). The parser file is in the PSDLGrammar.jj in Appendix C. The grammar rule that are used to create the parser is show in Appendix A.

7. Package caps.Psd1

The Psd1 package contains the classes that implement the data structure used to represent the PSDL prototype. This design has been modified to: (1) add the required data members and functions for "undo and redo" design, (2) add the required data members and functions for real time design, (3) modify the items related to the changes on the PSDLParser.jj and PSDLBuilder.jj, and (4) testing and debugging previous package.

V. SUMMARY AND CONCLUSION

A. USER INTERFACE EVALUATION

1. Correctness

After extensive testing, the editor now functions correctly. The user interface works gracefully and meets the users needs.

2. Easy to use

This implementation has made the editing operations intuitive. It becomes very easy for the user to figure out how to operate this new portable PSDL editor.

3. High productivity

Using Java JSwing components provide a good look & feel user interface. One can use them to their utmost advantage for the user by studying the user's operative behavior and build an efficient navigational structure accordingly. In this implementation, each single operation such as add, delete, decompose a vertex, change size/shape of data flow component will require only one mouse click. More demanding operations only need one or two mouse click. Every graphic symbol in the portable PSDL editor is readily understandable. There is no confusion among tree, button, menu, and dialog. Working with JTree is much easier and more efficient than ever before.

4. Automatic Error Detection

There is an automatic error detection for the user's input, which are checked by a grammar enforcer. After discovering a grammatical error, a pop-up message will be displayed with specific information to help the user to correct the error.

B. FUNCTIONAL EVALUATION

1. Stability and Portability

Stability is the heart of PSDL editor. The new portable PSDL editor has been working well. It can generate a correct version of PSDL code and convert PSDL code to graphic symbol with no error message. This portable editor is written in pure Java code, so it can run on any Java platforms.

2. Easy to Work With

This new portable PSDL editor has been enhanced with multi-level Undo and Redo mechanism. This makes undo and redo complex PSDL graphic symbols between different levels of the design feasible.

The automatic vertex and edge property consistency enforcer makes design job error free and increases the productivity of the user.

C. CONCLUSION

1. Results Of This Research

In this research, the new Java based PSDL editor has

been shown successful in its functionality and more valuable than ever. This new editor has been used in classroom as a tool for students to design their computer-aided prototyping project.

2. Future Work

An embedded real time software system must satisfy a set of real time constraints related to property and performance. This PSDL editor provides a basic local timing check. The function for the global timing constraints consistency check is required in the future design.

THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX A. PSDL GRAMMAR

The following is the complete specification of the Prototype System Description Language (PSDL) syntax extended Backus-Naur Form (BNF).

The BNF description of PSDL specifies the sequence of symbols, which consolidate a valid PSDL prototype. BNF describes the language in terms of production rules. Each production rule equates a non-terminal symbol to a sequence of terminal and non-terminal symbols. Terminal symbols are symbols, which can occur in PSDL. Non-terminal symbols are metalinguistic variables whose value is a sequence of symbols, which represent a PSDL construct.

Terminals are represented as **bold** symbols. Non-terminals are enclosed in angle brackets, < and >. Additional metasymbols are introduced in the extension of BNF to reduce the number of productions and non-terminals. These metasymbols are defined as:

- Square brackets, [] , to enclose optional items.
- Curly braces, { } , to enclose items which may appear zero or more times.
- Vertical bars, | , to represent a choice between items.
- Parentheses, () , to represent a grouping of items.

In some cases, the metasymbols are also used as terminals within PSDL. In order to avoid confusion, such terminal symbols are enclosed within single quotes.

For ease of reference, each production rule is numbered on the left hand side. These numbers are not part of the PSDL syntax.

1. <psdl>

::= { <component> }

2. <component>

::= <data_type>

| <operator>

3. <data_type>

::= **type** <id> <type_spec> <type_impl>

4. <type_spec>

- ::= specification** [**generic** < type_decl >] [< type_decl >
 { **operator** < op_name > < operator_spec > }
 [< functionality >] **end**
5. < operator >
- ::= operator** < op_name > < operator_spec > < operator_impl >
- 6 < operator_spec >
- ::= specification** { < interface > } [< functionality >] **end**
7. < interface >
- ::=** < attribute > [< reqmts_trace >]
8. < attribute >
- ::= generic** < type_decl >
 | **input** < type_decl >
 | **output** < type_decl >
 | **states** < type_decl > **initially** < initial_expression_list >
 | **exceptions** < id_list >
 | **maximum execution time** < time >
9. < type_decl >
- ::=** < id_list > : < type_name > { , < id_list > : < type_name > }
10. < type_name >
- ::=** < id >
 | < id > ' [' < type_decl > '] '
11. < id_list >
- ::=** < id > { , < id > }
12. < reqmts_trace >
- ::= required by** < id_list >
13. < functionality >
- ::=** [< keywords >] [< informal_desc >] [< formal_desc >]
14. < keywords >
- ::= keywords** < id_list >

15. < informal_desc >

::= **description** '{' < text > '}'

16. < formal_desc >

::= **axioms** '{' < text > '}'

17. < type_impl >

::= **implementation** < id > < id > **end**

| **implementation** < type_name >

{ **operator** < op_name > < operator_impl > } **end**

18. < operator_impl >

::= **implementation** < id > < id > **end**

| **implementation** < psdl_impl > **end**

19. < psdl_impl >

::= < data_flow_diagram > [< streams >] [< timers >]

[< control_constraints >] [< informal_desc >]

20. < data_flow_diagram >

::= **graph** { < vertex > } { < edge > }

21. < vertex >

::= **vertex** < op_id > [: < time >] { < property > }

22. < edge >

::= **edge** < id > [: < time >] < op_id > \longrightarrow < op_id > { < property > }

23. < property >

::= **property** < id > = < expression >

24. < op_id >

::= [< id > .] < op_name > ['(' [< id_list >] ' | ' [< id_list >] ')' ']

25. < streams >

::= **data_stream** < type_decl >

26. < timers >

::= **timer** < id_list >

27. < control_constraints >

::= control constraints < constraint > { < constraint > }

28. < constraint >

::= operator < op_id >
[**triggered** [< trigger >] [**if** < expression >] [< reqmts_trace >]]
[**period** < time > [< reqmts_trace >]]
[**finish within** < time > [< reqmts_trace >]]
[**minimum calling period** < time > [< reqmts_trace >]]
[**maximum response time** < time > [< reqmts_trace >]]
{ < constraint_options > }

29. < constraint_options >

::= output < id_list > **if** < expression > [< reqmts_trace >]
| **exception** < id > [**if** < expression >] [< reqmts_trace >]
| < timer_op > < id > [**if** < expression >] [< reqmts_trace >]

30. < trigger >

::= by all < id_list >
| **by some** < id_list >

31. < timer_op >

::= reset timer
| **start timer**
| **stop timer**

32. < initial_expression_list >

::= < initial_expression > { , < initial_expression > }

33. < initial_expression >

::= true
| **false**
| < integer_literal >
| < real_literal >
| < string_literal >
| < id >

```

| < type_name > . < op_name > [ '(' < initial_expression_list > ')' ]
| '(' < initial_expression > ')'
| < initial_expression > < binary_op > < initial_expression >
| < unary_op > < initial_expression >

```

34. < binary_op >

```

::= and | or | xor
| < | > | = | >= | <= | /=
| + | - | & | * | / | mod | rem | **

```

35. < unary_op >

```

::= not | abs | - | '+'

```

36. < time >

```

::= < integer_literal > < unit >

```

37. < unit >

```

::= microsec | ms | sec | min | hours

```

38. < expression_list >

```

::= < expression > { , < expression > }

```

39. < expression_list >

```

::= true
| false
| < integer_literal >
| < time >
| < real_literal >
| < string_literal >
| < id >
| < type_name > . < op_name > [ '(' < expression_list > ')' ]
| '(' < expression > ')'
| < expression > < binary_op > < expression >
| < unary_op > < expression >

```

40. < op_name >

- ::= < id >
41. < id >
- ::= < letter > { < alpha_numeric > }
42. < real_literal >
- ::= < integer_literal > . < integer_literal >
43. < integer_literal >
- ::= < digit > { < digit > }
44. < string_literal >
- ::= “ { < char > } “
45. < char >
- ::= any printable character except ‘ ’
46. < digit >
- ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
47. < letter >
- ::= a | b | c | d | e | f | g | h | i | j | k | l
 | m | n | o | p | q | r | s | t | u | v | w | x
 | y | z | A | B | C | D | E | F | G | H | I | J
 | K | L | M | N | O | P | Q | R | S | T | U
 | V | W | X | Y | Z
48. < alpha_numeric >
- ::= < letter >
 | < digit >
 | ‘ _ ’
49. < text >
- ::= { < char > }

APPENDIX B. ADA TEMPLATE GRAMMAR

1. `<ada_template>` ::= `<statement><package>`
2. `<statement>` ::= `with <root_name> + _instantiations;`
`use <root_name> + _instantiations;`
`with <root_name> + _exceptions;`
`use <root_name> + _exceptions;`
3. `<package>` ::= `<package_is><package_body>`
4. `<package_is>` ::= `package <name> + _pkg is`
`procedure <name> (<edge>)`
`end <name> + _pkg;`
5. `<package_body>` ::= `package <name> + _pkg is`
`procedure <name> (<edge>) is`
`begin`
`null; —your implementation goes here`
`end <name> ;`
`end <name> + _pkg;`
6. `<root_name>` ::= `<id> _ <digit>`
7. `<name>` ::= `<id> _ <digit>`
8. `<edge>` ::= `{ <id> : <edge_type> }`
- `<edge_type>` ::= `in <type>`
`| out <type>`
`| in out <type>`
9. `<type>` ::= `<id>`
10. `<id>` ::= `< string > { _ <string> }`
11. `<string>` ::= `A to Z`
`| its combination`
12. `<digit>` ::= `0 to 9`
`| its combination`

“+” use to concatenate two strings

THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX C. 1.PSDLGrammar.jj

```
/**
 * Program : PsdlGrammar.jj
 * Author   : Shen-Yi Tao
 * Version  : 1.1
 * This grammar file is written in JavaCC version 1.1
 */

options {
    IGNORE_CASE = true;          // PSDL is not case sensitive
    DEBUG_PARSER = true;         // Set this flag to true to trace the
    parser calls
}

PARSER_BEGIN (PsdlParser)

package caps.Parser;

import java.io.*;
import caps.Psdl.*;
import java.util.Vector;

public class PsdlParser {

    public static void main (String args[]) throws ParseException {
    }
} // End of the class PsdlParser

PARSER_END (PsdlParser)

/* White Space */
SKIP :
{
    " "
|   "\r"
|   "\t"
|   "\n"
}

/* Reserved Words */
TOKEN :
{
    < IF : "if" >
|   < MS : "ms" >
|   < SEC : "sec" >
|   < END : "end" >
|   < MIN : "min" >
|   < TYPE : "type" >
|   < EDGE : "edge" >
|   < TRUE : "true" >
|   < FALSE : "false" >
|   < GRAPH : "graph" >
|   < TIMER : "timer" >
|   < HOURS : "hours" >
|   < INPUT : "input" >
|   < PERIOD : "period" >
|   < STATES : "states" >
|   < AXIOMS : "axioms" >
|   < OUTPUT : "output" >
|   < VERTEX : "vertex" >
}
```

```

    < GENERIC : "generic" >
    < MICROSEC : "microsec" >
    < OPERATOR : "operator" >
    < KEYWORDS : "keywords" >
    < PROPERTY : "property" >
    < TRIGGERED : "triggered" >
    < EXCEPTION : "exception" >
    < INITIALLY : "initially" >
    < EXCEPTIONS : "exceptions" >
    < DESCRIPTION : "description" >
    < SPECIFICATION : "specification" >
    < IMPLEMENTATION : "implementation" >
    < NETWORKMAPPING : "netwrok_mapping" >
}

/* Operators */
TOKEN :
{
/* Binary Operators */
    < OR : "or" >
    < AND : "and" >
    < MOD : "mod" >
    < REM : "rem" >
    < XOR : "xor" >
    < GREATER_THAN : ">" >
    < LESS_THAN : "<" >
    < EQUALS : "=" >
    < GREATER_OR_EQUAL_TO : ">=" >
    < LESS_OR_EQUAL_TO : "<=" >
    < DIVIDE_EQUALS : "/=" >
    < PLUS : "+" >
    < MINUS : "-" >
    < AMPERCENT : "&" >
    < STAR : "*" >
    < FACTOR : "/" >
    < STAR_STAR : "***" >
/* Unary Operators */
    < ABS : "abs" >
    < NOT : "not" >
}

/* String real, and integer literals */
TOKEN :
{
    < TEXT : "{" ( < CHAR_TEXT > )* "}" >
    < #CHAR_TEXT : ~["}"] >

    < STRING_LITERAL : "\"" ( < CHAR_LIT > )* "\"" >
    < #CHAR_LIT : ~["}","\""] >

    < INTEGER_LITERAL : < INT_DIGIT > ( < INT_DIGIT > )* >
    < #INT_DIGIT : ["0" - "9"] >
}

// delete this for it can not prevent double "_" and can end with "_"
//TOKEN :
//{
//    <IDENTIFIER : < ID_LETTER > ( < ID_LETTER > | < ID_DIGIT > | "_" )*
//    >
//    | < #ID_LETTER : ["a" - "z"] | ["A" - "Z"] >

```

```

// | < #ID_DIGIT : ["0" - "9"] >
//}

/**
 * prevent double "_" and insure begin with letter
 * and end with letter or digit
 */
TOKEN :
{
    < IDENTIFIER : < ID_LETTER > ( (<DASH>)? < LETTERORDIGIT > )* >

    | < #LETTERORDIGIT : <ID_LETTER> | <ID_DIGIT> >
    | < #DASH : ["_"] >
    | < #ID_LETTER : ["a" - "z"] | ["A" - "Z"] >
    | < #ID_DIGIT : ["0" - "9"] >
}

/* Digits */
TOKEN :
{
    < DIGIT : ["0" - "9"] >
    | < LETTER : ["a" - "z"] | ["A" - "Z"] >
}

/* network name */
TOKEN :
{
    < STR : < STRING_LITERAL > >
}

/**
 * Production 1
 */
void psdl () :
{}
{
    ( component () ) *
}

/**
 * Production 2
 */
void component () :
{}
{
    data_type ()
    | operator ()
}

/**
 * Production 3
 * 11/9/99
 * change this as output file
 */
void data_type () :
{}
{
    <TYPE> <IDENTIFIER> type_spec () type_impl ()
}

/**

```

```

* Production 4
* <functionality> is directly included in this production
*/
void type_spec() :
{}
{
    <SPECIFICATION> [ <GENERIC> type_decl () ] [ type_decl () ]
    ( <OPERATOR> op_name () operator_spec () ) *
    [ keywords () ] [informal_desc () ] [ formal_desc () ] <END>
}

/**
* Production 5
* change this as output file
*/
void operator () :
{}

{
    <OPERATOR> < IDENTIFIER > [ "." < IDENTIFIER > ] [ "(" <
IDENTIFIER > "|"
    < IDENTIFIER > ")" ] [ operator_spec () operator_impl
() ]
}

/**
* Production 6
* <functionality> is directly included in this production
*/
void operator_spec () :
{}
{
    <SPECIFICATION> ( inter_face () ) * [ keywords () ] [informal_desc
() ] [ formal_desc () ] <END>
}

/**
* Production 7
*/
void inter_face () :
{}
{
    attribute () [ reqmts_trace () ]
}

/**
* Production 8
*/
void attribute () :
{}
{
    <GENERIC> type_decl ()
    | <INPUT> type_decl ()
    | <OUTPUT> type_decl ()
    | <STATES> type_decl () <INITIALLY> initial_expression_list ()
    | <EXCEPTIONS> id_list ()
    | "maximum execution time" time ()
}

/**
* Production 9

```

```

    */
void type_decl () :
{
{
    id_list () ":" type_name () ( "," id_list () ":" type_name () ) *
}

/**
 * Production 10
 * This production is modified to remove common prefix id ()
 */
//void type_name () :
//{}
//{
//    op_name ()
//    | op_name () "[" type_decl () "]"
//}
//

void type_name () :
{
{
    op_name () type_name_suffix ()
}

void type_name_suffix () :
{
{
    "[" type_decl () "]"
    | empty_string ()
}

/**
 * Production 11
 */
void id_list () :
{
{
    op_name () ( "," op_name () ) *
}

/**
 * Production 12
 */
void reqmts_trace () :
{
{
    "required by" id_list ()
}

/**
 * Production 13
 */
/* This production is included directly in other productions,
   because it caused empty string
void functionality () :
{
{
    [ keywords () ] [informal_desc () ] [ formal_desc () ]
}
*/

```

```

/**
 * Production 14
 */
void keywords () :
{
    <KEYWORDS> id_list ()
}

/**
 * Production 15
 */
void informal_desc () :
{
    <DESCRIPTION> < TEXT >
}

/**
 * Production 16
 */
void formal_desc () :
{
    <AXIOMS> < TEXT >
}

/**
 * Production 17
 * change this to not allow "_" and reduce lookahead
 */
void type_impl () :
{
    <IMPLEMENTATION> op_name () type_impl_suffix ()
}

/**
 * Production 11
 * change this to not allow "_" and reduce lookahead
 * This production is to remove the common prefix "implementation"
 */
void type_impl_suffix () :
{
    op_name () <END>
    | [ "[" type_name_suffix () "]" ] ( <OPERATOR> op_name ()
operator_impl () ) * <END>
}

/**
 * Production 18
 * change this to not allow "_" and reduce lookahead
 */
void operator_impl () :
{
    <IMPLEMENTATION> operator_impl_suffix ()
}

```



```

/**
 * change this to not allow "_" and reduce lookahead
 * This production is to remove common prefix "implementation"
 */
void operator_impl_suffix () :
{}
{
    op_name () op_name () <END>
    | psdl_impl () <END>
}

/**
 * Production 19
 */
void psdl_impl () :
{}
{
    data_flow_diagram () [ streams () ][ timers () ]
    [ control_constraints () ] [ informal_desc () ]
}

/**
 * Production 20
 */
void data_flow_diagram () :
{}
{
    <GRAPH> ( vertex () ) * ( edge () ) *
}

/**
 * Production 21
 * this allows two main types input from user
 * reduce look ahead
 */
/*void vertex () :
{}
{
    <VERTEX> < IDENTIFIER > vertex_type() [ ":" time () ] ( property
() ) *

}*/
void vertex () :
{}
{
    <VERTEX> < IDENTIFIER > [vertex_type()] [ ":" time () ] ( property
() ) *

}

/**
 * Production 22
 */
/*void edge () :
{}
{
    <EDGE> < IDENTIFIER > [ ":" time () ] ( < LETTER > ) *
    [<INTEGER_LITERAL>] vertex_type() "->" ( < LETTER > ) *
    [<INTEGER_LITERAL>]
    vertex_type() ( property () ) *

```

```

}
*/

void edge () :
{}
{
    <EDGE> < IDENTIFIER > [ ":" time () ] <IDENTIFIER>
    [vertex_type()] "->" <IDENTIFIER> [vertex_type()] ( property ()
)*
}

/**
 * Production 23
 */
void property () :

{}

{
    <PROPERTY> op_name () "=" expression ()
}

/**
 * Production 24
 * delete op_id
 * vertex_type() make this grammar judge and accept two_type inputs.
(id
 * ,op_id)
 * use look ahead technology :The job of a parser is to read an input
stream
 * and determine whether or not the input stream conforms to the
grammar.
 */

void vertex_type() :
{}
{
    "." <IDENTIFIER> [ "(" [ id_list () ] "|" [ id_list () ] ")" ]

// | "_" <INTEGER_LITERAL> ["_" <INTEGER_LITERAL>]
}

/**
 * Production 25
 */
void streams () :
{}
{
    "data stream" type_decl ()
}

/**
 * Production 26
 */
void timers () :
{}

{
    <TIMER> id_list ()
}

```

```

/**
 * Production 27
 */
void control_constraints () :
{
    "control constraints" constraint () ( constraint () ) *
}

/**
 * Production 28
 */
/*void constraint () :
{
    <OPERATOR> < IDENTIFIER > vertex_type()

    [ <TRIGGERED> [ trigger () ] [ <IF> expression () ] [ reqmts_trace
() ] ]
    [ <PERIOD> time () [ reqmts_trace () ] ]
    [ "finish within" time () [ reqmts_trace () ] ]
    [ "minimum calling period" time () [ reqmts_trace () ] ]
    [ "maximum response time" time () [ reqmts_trace () ] ]
    ( constraint_options () ) *
}*/
void constraint () :
{
    <OPERATOR> < IDENTIFIER > [ vertex_type() ]

    [ <TRIGGERED> [ trigger () ] [ <IF> expression () ] [ reqmts_trace
() ] ]
    [ <PERIOD> time () [ reqmts_trace () ] ]
    [ "finish within" time () [ reqmts_trace () ] ]
    [ "minimum calling period" time () [ reqmts_trace () ] ]
    [ "maximum response time" time () [ reqmts_trace () ] ]
    ( constraint_options () ) *
}

/**
 * Production 29
 */
void constraint_options () :
{
    <OUTPUT> id_list () <IF> expression () [ reqmts_trace () ]
    | <EXCEPTION> op_name () [ <IF> expression () ] [ reqmts_trace () ]
    | timer_op () op_name () [ <IF> expression () ] [ reqmts_trace () ]
}

/**
 * Production 30
 */
void trigger () :
{
    "by all" id_list ()
    | "by some" id_list ()
}

```

```

/**
 * Production 31
 */
void timer_op () :
{
    "reset timer"
    | "start timer"
    | "stop timer"
}

/**
 * Production 32
 */
void initial_expression_list () :
{
    initial_expression () ( "," initial_expression () ) *
}

/**
 * Production 33
 * This production has two common prefix problems and a left
recursion problem
 * and is modified .
 */
void initial_expression () :
{
    initial_expression_1 () initial_expression_tail ()
}

/**
 * change this to not allow "_" and decrease lookahead
 */

void initial_expression_1 () :
{
    < TRUE >
    | < FALSE >
    | < STRING_LITERAL >
    | < INTEGER_LITERAL > initial_expression_suffix1 ()
    | op_name () initial_expression_suffix2 ()
    | "(" initial_expression () ")"
    | unary_op () initial_expression ()
}

void initial_expression_tail () :
{
    binary_op () initial_expression () initial_expression_tail ()
    | empty_string ()
}

void initial_expression_suffix1 () :
{
    "." < INTEGER_LITERAL >
    | empty_string ()
}

```

```

}

void initial_expression_suffix2 () :
{
    [ "[" type_name_suffix () "]" ] "." op_name () [ "("
initial_expression_list () ")" ]
    | empty_string ()
}

/**
 * Production 34
 */
void binary_op () :
{
    <AND> | <OR> | <XOR> | <GREATER_THAN> | <LESS_THAN>
    | <EQUALS> | <GREATER_OR_EQUAL_TO> | <LESS_OR_EQUAL_TO>
    | <DIVIDE_EQUALS> | <PLUS> | <MINUS> | <AMPERCENT>
    | <STAR> | <FACTOR> | <MOD> | <REM> | <STAR_STAR>
}

/**
 * Production 35
 */
void unary_op () :
{
    <NOT> | <ABS> | <MINUS> | <PLUS>
}

/**
 * Production 36
 */
void time () :
{
    < INTEGER_LITERAL > unit ()
}

/**
 * Production 37
 */
void unit () :
{
    <MICROSEC>
    | <MS>
    | <SEC>
    | <MIN>
    | <HOURS>
}

/**
 * Production 38
 */
void expression_list () :
{
    expression () ( "," expression () ) *
}

```

```

/**
 * Production 39
 */
/** This production has two common prefix problems and a left
recursion problem and is modified */
void expression () :
{
    <TRUE>
    | <FALSE>
    | time ()
    | < INTEGER_LITERAL >
    | < REAL_LITERAL >
    | < STRING_LITERAL >
    | op_name ()
    | type_name () "." op_name () [ "(" expression_list () ")" ]
    | "(" expression () ")"
    | expression () binary_op () expression ()
    | unary_op () expression ()
}
*/

void expression () :
{
    expression_1 () expression_tail ()
}
/**
 * expression
 */

void expression_1 () :
{
    < TRUE >
    | < FALSE >
    | < STRING_LITERAL >
    | < INTEGER_LITERAL > expression_suffix1 ()
    | op_name () expression_suffix2 ()
    | "(" expression () ")"
    | unary_op () expression ()
    | network_mapping()
}

void expression_tail () :
{
    binary_op () expression () expression_tail ()
    | empty_string ()
}

void expression_suffix1 () :
{
    LOOKAHEAD (2)
    | "." < INTEGER_LITERAL >
    | unit ()
    | empty_string ()
}

```

```

}

void expression_suffix2 () :
{
    [ "[" type_name_suffix () "]" ] "." op_name () [ "(" expression_list
    () )" " ]
    | empty_string ()
}

/**
 * Production 40
 */
void op_name () :
{
    < IDENTIFIER >
}

/**
 * Production 41
 */
/*void id () :
{
    < IDENTIFIER >
}
*/
/*
 * Production 42
 */
/* This is a token, it is removed from the parser for efficiency
void real_literal () :
{
    < INTEGER_LITERAL > "." < INTEGER_LITERAL >
}
*/

/**
 * Production 43
 */
void integer_literal () :
{
    < INTEGER_LITERAL >
}

/*
 * Production 44
 */
/* This is a token, it is removed from the parser for efficiency
void string_literal () :
{
    < STRING_LITERAL >
}
*/

/*
 * Production 45

```

```

*/
/* This is a token, it is removed from the parser for efficiency
void digit () :
{}
{
    < DIGIT >
}
*/

/*
    Production 46
*/
/* This is a token, it is removed from the parser for efficiency
void letter () :
{}
{
    < LETTER >
}
*/

/*
    Production 47
*/
/* This goes into < IDENTIFIER >, it is removed from the parser
void alpha_numeric () :
{}
{
    letter ()
    | digit ()
    | "_"
}
*/

/*
    * Production 48
    */
/* This production goes into < TEXT > */
/*
void text () :
{}
{
    < TEXT >
}
*/

/**
    * Production 49
    */
/** Represents the empty string, not a part of the PSDL grammar */
void empty_string () :
{}
{
    { return; }
}

/**
    * Production 50
    */
void network_mapping () :

```



```

{}

{
    <NETWORKMAPPING> "=" <STR>
}

/**
 * This production is not in the grammar
 * It is used to check output guards of a vertex
 */
void check_output_guards () :
{
}
{
    ( < OUTPUT > id_list () < IF > expression () [reqmts_trace ()] )+
}

/**
 * This production is not in the grammar
 * It is used to check exception guards of a vertex
 */
void check_exception_guards () :
{
}
{
    ( < EXCEPTION > op_name () [ < IF > expression () ] [reqmts_trace
    ()] )+
}

/**
 * This production is not in the grammar
 * It is used to check exception list of a vertex
 */
void check_exception_list () :
{
}
{
    ( < EXCEPTIONS > id_list () )+
}

/**
 * This production is not in the grammar
 * It is used to check timer ops of a vertex
 */
void check_timer_ops () :
{
}
{
    ( timer_op () op_name () [ < IF > expression () ] [reqmts_trace
    ()] )+
}

/**
 * This production is not in the grammar
 * It is used to check parent specs
 */
void check_parent_spec () :
{
}
{
    < OPERATOR > op_name () operator_spec ()
}

/**
 * add this to check input format. E.g. such as "operator_1" and
 * "Machine.process(inputArg|outputArg)"

```

```

    * use token to reduce lookahead
    */
void OP_ID() :
{
    < IDENTIFIER > [ "." < IDENTIFIER > ] [ "(" paramater() "]"
paramater() ")" ]
}

void paramater() :
{
    [ < IDENTIFIER > ( "," <IDENTIFIER>)*]
}

/**
 * add this to check input format. E.g. "operator1"
 * use token to reduce look ahead
 */
void ID() :
{
    {
        < IDENTIFIER >
    }
}

```

APPENDIX C. 2.PsdlBuilder.jj

```
/*
  Program : PsdlBuiler.jj
  Author  : Shen-Yi Tao modified from Ilker Duranlioglu's version
  This grammar file is written in JavaCC version 1.1
*/

options {
  IGNORE_CASE = true;          /* PSDL is not case sensitive*/
  DEBUG_PARSER = true;         /* Set this flag to true to
  trace the parser calls */
}

PARSER_BEGIN (PsdlBuilder)

package caps.Builder;

import java.io.*;
import java.util.*;
import caps.Psdl.*;
import javax.swing.tree.DefaultMutableTreeNode;

public class PsdlBuilder
{
  private static boolean isType =false;

  private static Vector dfcVector;

  private static Vector streamsVector;

  private static Vertex currentOp;

  private static Vertex currentChild;

  private static Edge currentEdge;

  private static Vector idList = new Vector ();

  private static int index;

  public static void main (String args[]) throws ParseException
  {
  }

  public static Vertex buildPrototype (File file)
  {
    BufferedReader reader = null;
    try
    {
      reader = new BufferedReader (new FileReader (file));
    }
    catch (FileNotFoundException e)
    {
      System.out.println (e);
    }
    dfcVector = new Vector ();
    streamsVector = new Vector ();
    idList = new Vector ();
  }
}
```

```

        currentOp = null;
        currentEdge = null;
        currentChild = null;
        //load file.
        ReInit (reader);
        try
        {
            //create psdl from file.
            psdl ();
        }
        catch (ParseException ex)
        {
            System.out.println (ex);
            System.exit (0);
        }
        Vertex root = findRoot ();

        if(root != null )
        {
            for(Enumeration e = root.children(); e.hasMoreElements() ;)
            {
                DefaultMutableTreeNode tempDFC =
                ( (DefaultMutableTreeNode)e.nextElement());
                if(tempDFC instanceof Vertex )
                {
                    if( !(dfcVector.contains(tempDFC) ) )
                        root.remove(tempDFC );
                }
            }
            dfcVector = null;
            idList = null;
            currentOp = null;
            currentEdge = null;
            currentChild = null;
            return root;
        }

        public static String label;

        public static int id;

        public static int idExtension;
        public static Vertex findOperator (String name, boolean
        doubleSuffix)
        {
            DataFlowComponent d;
            Vertex found = null;
            boolean isDotFound = false;
            extractLabel (name, doubleSuffix);
            if(name.indexOf(".") != -1)
            {
                isDotFound = true;
                doubleSuffix = false;
            }
            for (Enumeration enum = dfcVector.elements ();
            enum.hasMoreElements ());
            {
                d = (DataFlowComponent) enum.nextElement ();
                String str = "";

```

```

        if(isDotFound)
        {
            if(d.getLabel ().indexOf(".") != -1)
            {
                int leftbrace = d.getLabel ().indexOf("(");
                String tempStr = d.getLabel ();
                str = tempStr.substring(0,leftbrace);
                str = str.concat("_" + d.getId() +

                tempStr.substring(leftbrace,tempStr.length() ) );
                if ((d instanceof Vertex) && (str.equals (name)))
                    found = (Vertex) d;
            }
        }
        else if (doubleSuffix)
            str = new String (d.getLabel () + "_" + d.getId () + "_"

            +(((Vertex) d).getIdExtension ());
        else
            str = new String (d.getLabel () + "_" + d.getId ());
        if ((d instanceof Vertex) && (str.equals (name)))
            found = (Vertex) d;
    }
    //new vector to be added to vector.
    if (found == null)
    {
        if (doubleSuffix == false & !isDotFound)
            found = new Vertex (0, 0, null, false);    // This is the
                                                    // root

        else if(isDotFound)
            found = new Vertex (0, 0, currentOp, false);    // child
                                                    //vertex

        else
            found = new Vertex (0, 0, currentOp, false);    //child
                                                    //vertes

        found.setLabel (label);
        found.setId (id);
        found.setIdExtension (idExtension);
        dfcVector.addElement (found);
    }
    else if( doubleSuffix && found.getParent () == null )
    {
        currentOp.add (found);
    }
    else if( isDotFound && found.getParent () == null )
    {
        currentOp.add (found);
    }
    }

    return found;
}

public static void extractLabel (String s, boolean doubleSuffix)
{
    boolean isDotFound = false;
    String str="";
    int leftParenthesis = -1;

    if(s.indexOf(".") == -1)
    {
        isDotFound = false;

```

```

    }
    else
    {
        isDotFound = true;
        if( (s.indexOf( "(" ) ) != -1 )
        {
            leftParenthesis = s.indexOf("(");
            str = s.substring (0, leftParenthesis);
        }
        else
        {
            System.out.println("error!can't find the" + "(" + " in
                the PsdlBuilder.extractLabel()" );
        }
    }
}
// machine.process(|)
if(isDotFound)
{
    int index = s.lastIndexOf ("_");
    String temp = str.substring (index + 1, str.length ());
    int num = Integer.parseInt (temp);
    str = new String (s.substring (0, index)
        + s.substring(leftParenthesis, s.length()) );
    label = str;
    id = num;
}
//operator_1
else
{
    int index = s.lastIndexOf ("_");
    String temp = s.substring (index + 1, s.length ());
    int num = Integer.parseInt (temp);
    s = new String (s.substring (0, index));
    if (doubleSuffix == false) // If contains only one suffix
    {
        label = s;
        id = num;
    }
    else
    {
        idExtension = num;
        index = s.lastIndexOf ("_");
        temp = s.substring (index + 1, s.length ());
        num = Integer.parseInt (temp);
        s = new String (s.substring (0, index));
        label = s;
        id = num;
    }
}
}
}
/*
public static void extractLabel (String s, boolean doubleSuffix)
{
    int index = s.lastIndexOf ("_");
    //add flowing code to catch the first index and end index for
    //retrieve a number

    boolean hasProcess = false;
    int endIndex;

```

```

        if( (s.indexOf( "(" ) ) != -1 )
        {
            endIndex = s.indexOf("(");
            hasProcess = true;
        }
        else
        {
            endIndex = s.length();
            hasProcess = false;
        }
        //following code is error on id and has been modified.
        String temp = s.substring (index + 1, s.length ());

        String temp = s.substring (index + 1, endIndex);
        int num = Integer.parseInt (temp);

        if (doubleSuffix == false) {    // If contains only one suffix
            if (hasProcess)
            {
                String labelString = s.substring(0,index );
                label=labelString.concat(s.substring(endIndex,s.length()))
            };
            id = num;
        }
        else
        {
            label = s.substring (0, index);
            id = num;
        }
    }
    else {
        idExtension = num;
        String labelString = s.substring(0,index);
        if (hasProcess)
        {
            label = labelString.concat( s.substring( endIndex,
                s.length()))
        };

        int tempStart = label.lastIndexOf("_");
        int tempEnd= label.indexOf("(");
        id = Integer.parseInt
        (label.substring(tempStart+1,tempEnd) );
    }
    else
    {
        label = labelString ;
        int tempStart = label.lastIndexOf("_");
        id = Integer.parseInt
            (label.substring(tempStart+1,label.length()) );
    }
}
}
*/
public static Edge findEdge (String name)
{
    DataFlowComponent d;
    Edge found = null;
    for (Enumeration enum = streamsVector.elements ())
enum.hasMoreElements
    ());

```

```

        {
            d = (DataFlowComponent) enum.nextElement ();
            if ((d instanceof Edge) && (d.getLabel ().equals (name)))
                found = (Edge) d;
        }
        return found;
    }

    public static Vertex findRoot ()
    {
        Vertex o = null;
        DataFlowComponent d;
        for (Enumeration enum = dfcVector.elements
());enum.hasMoreElements ());
        {
            d = (DataFlowComponent) enum.nextElement ();
            if (d.getParent () == null)
                o = (Vertex) d;
        }
        return o;
    }

    public static String extractIdList (Vector v)
    {
        String str = "";
        Enumeration enum;
        if (v != null) {
            enum = v.elements ();
            if (enum.hasMoreElements ())
                str = new String ((String) enum.nextElement ());
            while (enum.hasMoreElements ()) {
                str = str.concat (" ").concat ((String) enum.nextElement
());
            }
        }
        return str;
    }

    public static Vertex findChild (String name)
    {
        DataFlowComponent d;
        Vertex found = null;
        extractLabel (name, true); // DoubleSuffix
        for (Enumeration enum = currentOp.children ();
enum.hasMoreElements ()); {
            d = (DataFlowComponent) enum.nextElement ();
            if ((d instanceof Vertex) && (d.getLabel ().equals (label)))
                found = (Vertex) d;
        }
        return found;
    }

    public static void setCurrentOp (Vertex v)
    {
        currentOp = v;
    }

    public static void setVertexProperty (Vertex v, String prop,
String exp)
    {
        if (prop.equalsIgnoreCase ("x"))

```



```

        v.setX (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("y"))
        v.setY (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("radius"))
        v.setWidth (Integer.parseInt (exp) * 2);
    else if (prop.equalsIgnoreCase ("color"))
        v.setColor (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("label_font"))
        v.setLabelFontIndex (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("label_x_offset"))
        v.setLabelXOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("label_y_offset"))
        v.setLabelYOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("met_font"))
        v.setMetFontIndex (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("met_unit")) {
        if (v.getMet () != null)
            v.getMet ().setTimeUnits (exp);
    }
    else if (prop.equalsIgnoreCase ("met_x_offset"))
        v.setMetXOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("met_y_offset"))
        v.setMetYOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("is_terminator"))
    {
        if (exp.equalsIgnoreCase ("true"))
            v.setTerminator (true);
    }
    else if (prop.equalsIgnoreCase ("network_mapping"))
    {
        if (exp.length() == 2 )
            v.setNetWorkLabel("");
        else
        {
            String str = exp.substring(1, (exp.length()-1));
            v.setNetWorkLabel(str);
        }
    }
    else if (prop.equalsIgnoreCase ("criticalness"))
    {
        if (exp.equalsIgnoreCase("\"\"+\"hard\"+\"\"") )
            v.setCriticalStatus(1);
        else if (exp.equalsIgnoreCase("\"\"+\"soft\"+\"\"") )
            v.setCriticalStatus(2);
        else
        {
            v.setCriticalStatus(3);
            v.setTimingType(0);
        }
    }
}

}

public static void setEdgeProperty (Edge e, String prop, String
                                   exp)
{
    if (prop.equalsIgnoreCase ("id"))
        e.setId (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("label_font"))
        e.setLabelFontIndex (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("label_x_offset"))

```

```

        e.setLabelXOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("label_y_offset"))
        e.setLabelYOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("latency_font"))
        e.setMetFontIndex (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("latency_unit")) {
        if (e.getMet () != null)
            e.getMet ().setTimeUnits (exp);
    }
    else if (prop.equalsIgnoreCase ("latency_x_offset"))
        e.setMetXOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("latency_y_offset"))
        e.setMetYOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase ("spline"))
        e.setInitialControlPoints (exp);
}

} // End of the class PsdlBuilder

PARSER_END (PsdlBuilder)

```

```

/* White Space */

```

```

SKIP :

```

```

{
    " "
|  "\r"
|  "\t"
|  "\n"
}

```

```

/* Reserved Words */

```

```

TOKEN :

```

```

{
    < IF : "if" >
|  < MS : "ms" >
|  < SEC : "sec" >
|  < END : "end" >
|  < MIN : "min" >
|  < TYPE : "type" >
|  < EDGE : "edge" >
|  < TRUE : "true" >
|  < FALSE : "false" >
|  < GRAPH : "graph" >
|  < TIMER : "timer" >
|  < HOURS : "hours" >
|  < INPUT : "input" >
|  < PERIOD : "period" >
|  < STATES : "states" >
|  < AXIOMS : "axioms" >
|  < OUTPUT : "output" >
|  < VERTEX : "vertex" >
|  < GENERIC : "generic" >
|  < MICROSEC : "microsec" >
|  < OPERATOR : "operator" >
|  < KEYWORDS : "keywords" >
|  < PROPERTY : "property" >
|  < TRIGGERED : "triggered" >
|  < EXCEPTION : "exception" >
|  < INITIALLY : "initially" >
|  < EXCEPTIONS : "exceptions" >

```



```

{
    < IDENTIFIER  : < ID_LETTER > ( (<DASH>)? < LETTERORDIGIT > ) * >

    | < #LETTERORDIGIT : <ID_LETTER> | <ID_DIGIT>  >
    | < #DASH : ["_"] >
    | < #ID_LETTER : ["a" - "z"] | ["A" - "Z"] >
    | < #ID_DIGIT : ["0" - "9"] >
}

/* Digits and letters */
TOKEN :
{
    < DIGIT : ["0" - "9"] >
    | < LETTER : ["a" - "z"] | ["A" - "Z"] >
}

/* network name */
TOKEN :
{
    < STR : < STRING_LITERAL> >
}

/**
 * Production 1
 */
void psdl () :
{
{
    ( component () ) *
}
}

/**
 * Production 2
 */
void component () :
{
{
    data_type ()
    | operator ()
}
}

/**
 * Production 3
 * 11/9/99
 * change this as output file
 */
void data_type () :
{
{
    <TYPE> { isType = true; } <IDENTIFIER> type_spec () type_impl ()
    }
}

/**
 * Production 4
 */
/* <functionality> is directly included in this production */
void type_spec () :
{
{
    <SPECIFICATION> [ <GENERIC> type_decl (false) ] [ type_decl
(false) ]
}
}

```

```

    ( <OPERATOR> op_name () operator_spec () ) *
    [ keywords () ] [informal_desc () ] [ formal_desc () ] <END>
}

/**
 * Production 5
 */
void operator () :
{
    String name;
}
{
    <OPERATOR> {isType = false;}name = op_name ()
    {
        currentOp = findOperator (name, false);
    }
    operator_spec () operator_impl ()
    {
        currentOp = null;
    }
}

/**
 * Production 6
 */
/* <functionality> is directly included in this production */
void operator_spec () :
{
    String desc;
    Vector list;
}
{
    <SPECIFICATION> ( inter_face () ) *
    [ list = keywords () { currentOp.setKeywordList (list); } ]
    [ desc = informal_desc () { currentOp.setInformalDesc (desc); } ]
    [ desc = formal_desc () { currentOp.setFormalDesc (desc); } ]
    <END>
}

/**
 * Production 7
 */
void inter_face () :
{}
{
    attribute () /* reqmts_trace is under attribute */
}

/**
 * Production 8
 */
void attribute () :
{
    String initial;
    Token tok;
    Vector list;
    PSDLTime met;
    Vector reqmts = null;
    String str="";
}

```

```

{
    tok = <GENERIC> type_decl (false)
    {
        if(!isType){str = currentOp.getGenericList ();}
        if (str != "")
            str = str.concat ("\n");
        str = str.concat (tok.toString () + " " + extractIdList
(idList));
    }
    [ list = reqmts_trace () { str = str.concat ("\n  REQUIRED BY " +
extractIdList      (list)); } ]
    { if(!isType){currentOp.setGenericList (str); } }

    | <INPUT> type_decl (false) [ reqmts = reqmts_trace () ]
    { if(!isType){ ((Vector) currentOp.getSpecReqmts ().elementAt
(0)).addElement      (extractIdList (reqmts)); } }
    | <OUTPUT> type_decl (false) [ reqmts = reqmts_trace () ]
    { if(!isType){ ((Vector) currentOp.getSpecReqmts ().elementAt
(1)).addElement      (extractIdList (reqmts)); } }
    | <STATES> type_decl (true) <INITIALLY> initial =
initial_expression_list ()
    {
        currentEdge.setStateStream (true);
        currentEdge.setInitialValue (initial);
    }
    [ reqmts = reqmts_trace () ]
    { if(!isType){ ((Vector) currentOp.getSpecReqmts ().elementAt
(2)).addElement
        (extractIdList (reqmts)); } }
    | tok = <EXCEPTIONS> list = id_list ()
    { if(!isType)
        {
            if(!isType){str = currentOp.getExceptionList ();}
            if (str != "")
                str = str.concat ("\n");
            str = str.concat (tok.toString () + " " + extractIdList
(list));
        }
    }
    [ list = reqmts_trace () { str = str.concat ("\n  REQUIRED BY " +
extractIdList (list)); } ]
    { if(!isType){currentOp.setExceptionList (str);} }
    | "maximum execution time" met = time () {
if(!isType){currentOp.setMet (met);} }
    [ list = reqmts_trace () { if(!isType){currentOp.setMetReqmts
(list); } } ]
}

/**
 * bugs on type generations
 * Production 9
 */
void type_decl (boolean buildEdge) :
{
    Vector idList;
    String type = "";
}
{
    idList = id_list () ":" type = type_name ()
    {
        currentEdge = findEdge ((String) idList.elementAt (0));
    }
}

```

```

        if (buildEdge && (currentEdge == null))
        {
            currentEdge = new Edge (0, 0, currentOp);
            currentEdge.setLabel ((String) idList.elementAt (0));
            streamsVector.addElement (currentEdge);
        }
        if (buildEdge)
        {
            currentEdge.setStreamType (type);
            for (Enumeration e = streamsVector.elements() ;
e.hasMoreElements() ; )
            {
                Edge eg = (Edge)e.nextElement();
                currentEdge.getLabel().equals( eg.getLabel() );
                eg.setStreamType (type);
            }
        }
        ( "," idList = id_list () ":" type = type_name ()
        {
            if (buildEdge)
            {
                currentEdge = findEdge ((String) idList.elementAt (0));
                for (Enumeration enum = streamsVector.elements () ;
enum.hasMoreElements      ());
                {
                    Edge e = (Edge) enum.nextElement ();
                    if (currentEdge.getLabel ().equals (e.getLabel ()))
                        e.setStreamType (type);
                }
                for (Enumeration e = streamsVector.elements() ;
                    e.hasMoreElements() ; )
                {
                    Edge eg = (Edge)e.nextElement();
                    currentEdge.getLabel().equals( eg.getLabel() );
                    eg.setStreamType (type);
                }
            }
        }
    ) *
}

/**
 * Production 10
 */
/* This production is mofidied to remove common prefix id () */
/*
String type_name () :
{
    String name = "";
}
{
    name = id () { return name;}
| id () "[" type_decl (false) "]"
}
*/

String type_name () :
{
    String name;
}

```

```

{
    name = id () type_name_suffix ()
    { return name; }
}

/** This production is to remove the common prefix id () */
String type_name_suffix () :
{String str;}
{
    "[" type_decl (false) "]" { return ""; }
    | empty_string () { return ""; }
}

/**
 * Production 11
 */
Vector id_list () :
{
    idList = new Vector ();
    String name;
}
{
    name = id () { idList.addElement (name); }
    ( "," name = id () { idList.addElement (name); } ) *
    {return idList; }
}

/**
 * Production 12
 */
Vector reqmts_trace () :
{
    Vector list;
}
{
    "required by" list = id_list ()
    { return list; }
}

/**
 * Production 13
 */
/* This production is included directly in other productions,
   because it caused empty string
void functionality () :
{}
{
    [ keywords () ] [informal_desc () ] [ formal_desc () ]
}
*/

/**
 * Production 14
 */
Vector keywords () :
{
    Vector list;
}
{
    <KEYWORDS> list = id_list ()
    { return list; }
}

```



```

}

/**
 * Production 15
 */
String informal_desc () :
{
    Token tok;
    Token text;
}
{
    tok = <DESCRIPTION> text = < TEXT >
    { return new String (tok.toString () + " " + text.toString ()); }
}

/**
 * Production 16
 */
String formal_desc () :
{
    Token tok;
    Token text;
}
{
    tok = <AXIOMS> text = < TEXT >
    { return new String (tok.toString () + " " + text.toString ()); }
}

/**
 * Production 17
 */
/* This production is causing a common prefix problem and is modified
void type_impl () :
{}
{
    <IMPLEMENTATION> id () id () <END>
    | <IMPLEMENTATION> type_name ()
    ( <OPERATOR> op_name () operator_impl () ) * <END>
}
*/

void type_impl () :
{}
{
    <IMPLEMENTATION> id () type_impl_suffix ()
}

/** This production is to remove the common prefix "implementation"
*/
void type_impl_suffix () :
{}
{
    id () <END>
    | [ "[" type_name_suffix () "]" ] ( <OPERATOR> op_name ()
operator_impl () ) * <END>
}

/**
 * Production 18
 */

```

```

/* This production causes a common prefix problem and hence is
modified
void operator_impl () :
{
    <IMPLEMENTATION> id () id () <END>
    | <IMPLEMENTATION> psdl_impl () <END>
}
*/

void operator_impl () :
{
    <IMPLEMENTATION> operator_impl_suffix ()
}

/** This production is to remove common prefix "implementation" */
void operator_impl_suffix () :
{
    String language;
}
{
    language = id () id () <END>
    { currentOp.setImpLanguage (language); }
    | psdl_impl () <END>
}

/**
 * Production 19
 */
void psdl_impl () :
{
    String desc;
}
{
    data_flow_diagram () [ streams () ][ timers () ]
    [ control_constraints () ]
    [ desc = informal_desc () { currentOp.setGraphDesc (desc); } ]
}

/**
 * Production 20
 */
void data_flow_diagram () :
{
}
{
    <GRAPH> ( vertex () ) * ( edge () ) *
}

/**
 * Production 21
 */
void vertex () :
{
    String name;
    PSDLTime met;
}
{
    <VERTEX> name = op_id ()
    { currentChild = findOperator (name, true); }
}

```

```

    [ ":" met = time () { currentChild.setMet (met); } ] ( property
(currentChild) )*
    { currentChild = null;
    }
}

/**
 * Production 22
 */
void edge () :
{
    String name;
    PSDLTime latency;
    Vertex src;
    Vertex dest;
}
{
    <EDGE> name = id ()
    { if ((currentEdge = findEdge (name)) == null) {
        currentEdge = new Edge (0, 0, currentOp);
        currentEdge.setLabel (name);
        streamsVector.addElement (currentEdge);
    }
    else {
        if (currentEdge.getSource () == null) {
            streamsVector.removeElement (currentEdge);
            currentEdge.removeFromParent ();
        }
        Edge e = new Edge (0, 0, currentOp);
        e.setLabel (name);
        e.setStreamType (currentEdge.getStreamType ());
        e.setStateStream (currentEdge.isStateStream ());
        e.setInitialValue (currentEdge.getInitialValue ());
        streamsVector.addElement (e);
        currentEdge = e;
    }
}
[ ":" latency = time () { currentEdge.setMet (latency); } ]
name = op_id () { if (name.equals ("EXTERNAL")) {
    External ex = new External (0, 0, currentOp);
    ex.addOutEdge (currentEdge);
    currentEdge.setSource (ex);
}
else {
    src = findOperator (name, true);
    currentEdge.setSource (src);
    src.addOutEdge (currentEdge);
}
}
"->"
name = op_id () { if (name.equals ("EXTERNAL"))
{
    External ex = new External (0, 0, currentOp);
    ex.addInEdge (currentEdge);
    currentEdge.setDestination (ex);
}
else
{
    dest = findOperator (name, true);
    currentEdge.setDestination (dest);
    dest.addInEdge (currentEdge);
}
}

```

```

        }
    }
    ( property (currentEdge) ) *
}

/**
 * Production 23
 */
void property (DataFlowComponent dfc) :
{
    String prop;
    String exp;
}
{
    <PROPERTY> prop = id () "=" exp = expression ()
    { if (dfc instanceof Vertex)
        setVertexProperty ((Vertex) dfc, prop, exp);
      else
        setEdgeProperty ((Edge) dfc, prop, exp);
    }
}

/**
 * Production 24
 */
/* This production has common prefix problem and is modified
void op_id () :
{
    [ id () "." ] op_name () [ "(" [ id_list () ] "|" [ id_list () ]
    ")" ]
}
*/
/*
String op_id () :
{
    String name;
}
{
    name = op_name () [ "." id () ] [ "(" [ id_list () ] "|" [ id_list
    () ] ")" ]
    { return name; }
}*/
String op_id () :
{
    String str;
}
{
    str=vertex_type()
    {return str;}
}

/**
 * vertex_type() make this grammar judge and accept two_type inputs.
(id ,op_id)
 * use lookahead technology :The job of a parser is to read an input
stream
 * and determine whether or not the input stream conforms to the
grammar.
 */
String vertex_type() :

```

```

{
    String name;
    String name1;
    String name2;
    String name3;
    Vector idList1;
    Vector idList2;
}
{
    name = id()["." name1 = id()
    {name = name.concat("."+name1);} }
    ["("
    {name = name.concat("("); }
    [idList1=id_list()
    { name2 = extractIdList (idList1);
    name=name.concat(name2); }]
    "|"
    {name=name.concat("|");}
    [idList2=id_list()
    { name3 = extractIdList (idList2);
    name=name.concat(name3); }]
    ")"
    {name=name.concat(")");}]
    { return name;}
}

/**
 * Production 25
 */
void streams () :
{
{
    "data stream" type_decl (true)
    { streamsVector.removeAllElements (); }
}

/**
 * Production 26
 */
void timers () :
{
    Vector list;
}
{
    <TIMER> list = id_list () { currentOp.setTimerList (list); }
}

/**
 * Production 27
 */
void control_constraints () :
{
{
    "control constraints" constraint () ( constraint () ) *
}

/**
 * Production 28
 */
void constraint () :
{

```

```

    Vector list;
    String str;
    PSDLTime t;
}
{
    <OPERATOR> str = op_id ()
    { currentChild = findChild (str); }
    [ <TRIGGERED> [ list = trigger () { currentChild.setTriggerType
(index);

currentChild.setTriggerStreamsList (list); } ]
    [ <IF> str = expression () { currentChild.setIfCondition (str);
}]
    [ list = reqmts_trace () { currentChild.setTriggerReqmts
(list); } ] ]
    [ <PERIOD> t = time () { currentChild.setTimingType
(Vertex.PERIODIC);
                                currentChild.setPeriod (t); }
    [ list = reqmts_trace () { currentChild.setPeriodReqmts (list);
} ] ]
    [ "finish within" t = time () { currentChild.setFinishWithin (t);
}
    [ list = reqmts_trace () { currentChild.setFinishWithinReqmts
(list); } ]
    ]

    [ "minimum calling period" t = time () {
currentChild.setTimingType (Vertex.SPORADIC);
                                currentChild.setMcp (t);
}
    [ list = reqmts_trace () { currentChild.setMcpReqmts (list); }
] ]
    [ "maximum response time" t = time () { currentChild.setMrt (t); }
    [ list = reqmts_trace () { currentChild.setMrtReqmts (list); } ]
    ]
    ( constraint_options () ) *
}

/**
 * Production 29
 */
void constraint_options () :
{
    Token tok;
    Vector list;
    String expr = "";
    String str = "";
}
{
    tok = <OUTPUT> list = id_list () { str = new String (tok.toString
() + " " +
    extractIdList (list)); }
    tok = <IF> expr = expression () { str = new String (str + " " +
tok.toString () + " " + expr); }
    [ list = reqmts_trace () { str = new String (str + "\n    " +
"REQUIRED BY " +
    extractIdList (list)); } ]
    { currentChild.setOutputGuardList (str); }
    | tok = <EXCEPTION> expr = id () { str = new String (tok.toString ()
+ " " +
    expr); }
}

```

```

    [ tok = <IF> expr = expression () { str = new String (str + " " +
    tok.toString () + " " + expr); } ]
    [ list = reqmts_trace () { str = new String (str + "\n    " +
"REQUIRED BY " +
    extractIdList (list)); } ]
    { currentChild.setExceptionGuardList (str); }
    | str = timer_op () expr = id () { str = new String (str + " " +
expr); }
    [ tok = <IF> expr = expression () { str = new String (str + " " +
    tok.toString () + " " + expr); } ]
    [ list = reqmts_trace () { str = new String (str + "\n    " +
"REQUIRED BY " +
    extractIdList (list)); } ]
    { currentChild.setTimerOpList (str); }
}

/**
 * Production 30
 */
Vector trigger () :
{
    Vector list;
}
{
    "by all" list = id_list ()
    { index = 2;
      return list; }
    | "by some" list = id_list ()
    { index = 1;
      return list; }
}

/**
 * Production 31
 */
String timer_op () :
{}
{
    "reset timer" { return new String ("RESET TIMER"); }
    | "start timer" { return new String ("START TIMER"); }
    | "stop timer" { return new String ("STOP TIMER"); }
}

/**
 * Production 32
 */
String initial_expression_list () :
{
    String list = "";
    String expr = "";
}
{
    list = initial_expression () ( "," expr = initial_expression ()
    { list = list.concat (" , ").concat (expr); }
    ) *
    { return list; }
}

/**
 * Production 33
 */

```

```

/** This production has two common prefix problems and a left
recursion problem and is modified */
/*
void initial_expression () :
{}
{
    < TRUE >
    < FALSE >
    < INTEGER_LITERAL >
    < REAL_LITERAL >
    < STRING_LITERAL >
    id ()
    type_name () "." op_name () [ "(" initial_expression_list () ")" ]
    \
    "(" initial_expression () ")"
    initial_expression () binary_op () initial_expression ()
    unary_op () initial_expression ()
}
*/

String initial_expression () :
{
    String str = "";
    String tail = "";
}
{
    str = initial_expression_1 () tail = initial_expression_tail ()
    { return new String (str + tail); }
}

String initial_expression_1 () :
{
    Token tok;
    String str;
    String suffix = "";
}
{
    tok = < TRUE > { return tok.toString (); }
    tok = < FALSE > { return tok.toString (); }
    tok = < STRING_LITERAL > { return tok.toString (); }
    tok = < INTEGER_LITERAL > suffix = initial_expression_suffix1 ()
    { return new String (tok.toString () + suffix); }
    str = id () suffix = initial_expression_suffix2 ()
    { return new String (str + suffix); }
    "(" str = initial_expression () ")"
    { return new String "(" + str + ")"; }
    str = unary_op () suffix = initial_expression ()
    { return new String (str + suffix); }
}

String initial_expression_tail () :
{
    String str;
    String suffix1;
    String suffix2;
}
{
    str = binary_op () suffix1 = initial_expression () suffix2 =
initial_expression_tail ()
    { return new String (str + suffix1 + suffix2); }
    empty_string ()
}

```



```

    { return ""; }
}

String initial_expression_suffix1 () :
{
    Token tok;
}
{
    "." tok = < INTEGER_LITERAL >
    { return new String ( "." + tok.toString () ); }
    | empty_string ()
    { return ""; }
}

String initial_expression_suffix2 () :
{
    String str = "";
    String s = "";
}
{
    str = type_name_suffix () "." s = op_name ()
    { str = str.concat ( "." ).concat ( s ); }
    [ "(" s = initial_expression_list () ")" { str = new String ( str +
    "(" + s + ")" ); } ]
    { return str; }
    | empty_string ()
    { return ""; }
}

}

/**
 * Production 34
 */
String binary_op () :
{
    Token tok;
}
{
    tok = <AND> { return tok.toString (); }
    | tok = <OR> { return tok.toString (); }
    | tok = <XOR> { return tok.toString (); }
    | tok = <GREATER_THAN> { return tok.toString (); }
    | tok = <LESS_THAN> { return tok.toString (); }
    | tok = <EQUALS> { return tok.toString (); }
    | tok = <GREATER_OR_EQUAL_TO> { return tok.toString (); }
    | tok = <LESS_OR_EQUAL_TO> { return tok.toString (); }
    | tok = <DIVIDE_EQUALS> { return tok.toString (); }
    | tok = <PLUS> { return tok.toString (); }
    | tok = <MINUS> { return tok.toString (); }
    | tok = <AMPERCENT> { return tok.toString (); }
    | tok = <STAR> { return tok.toString (); }
    | tok = <FACTOR> { return tok.toString (); }
    | tok = <MOD> { return tok.toString (); }
    | tok = <REM> { return tok.toString (); }
    | tok = <STAR_STAR> { return tok.toString (); }
}

}

/**
 * Production 35
 */
String unary_op () :

```

```

{
    Token tok;
}
{
    tok = <NOT> { return tok.toString (); }
    tok = <ABS> { return tok.toString (); }
    tok = <MINUS> { return tok.toString (); }
    tok = <PLUS> { return tok.toString (); }
}

/**
 * Production 36
 */
PSDLTime time () :
{
    PSDLTime t = new PSDLTime ();
    String str = "";
    Token tok;
}
{
    tok = < INTEGER_LITERAL >
    { t.setTimeValue (Integer.parseInt (tok.toString ())); }
    str = unit ()
    { t.setTimeUnits (str); }
    { return t; }
}

/**
 * Production 37
 */
String unit () :
{
    Token tok;
}
{
    tok = <MICROSEC> { return tok.toString (); }
    tok = <MS> { return tok.toString (); }
    tok = <SEC> { return tok.toString (); }
    tok = <MIN> { return tok.toString (); }
    tok = <HOURS> { return tok.toString (); }
}

/**
 * Production 38
 */
String expression_list () :
{
    String expList = "";
    String str = "";
}
{
    expList = expression () ( "," str = expression ()
        { expList = expList.concat (" , ").concat (str); }
        ) *
    {
        return expList;
    }
}

/**
 * Production 39

```

```

    */
    /** This production has two common prefix problems and a left
    recursion problem and is modified */
    /*
    void expression () :
    {}
    {
        <TRUE>
        | <FALSE>
        | time ()
        | < INTEGER_LITERAL >
        | < REAL_LITERAL >
        | < STRING_LITERAL >
        | id ()
        | type_name () "." op_name () [ "(" expression_list () ")" ]
        | "(" expression () ")"
        | expression () binary_op () expression ()
        | unary_op () expression ()
    }
    */

    String expression () :
    {
        String exp = "";
        String expTail = "";
    }
    {
        exp = expression_1 () expTail = expression_tail ()
        {
            exp = exp.concat (expTail);
            return exp;
        }
    }

    String expression_1 () :
    {
        Token tok;
        String str = "";
        String suffix = "";
        PSDLTime t;
    }
    {
        tok = < TRUE > { return tok.toString (); }
        | tok = < FALSE > { return tok.toString (); }
        | tok = < STRING_LITERAL > { return tok.toString (); }
        | tok = < INTEGER_LITERAL > ( LOOKAHEAD (2) ( suffix =
    expression_suffix1 ()
    { return new String (tok.toString () + suffix); } ) |
    ( str = unit () { return new String (tok.toString () + str); } ) )
    | str = id () suffix = expression_suffix2 ()
    { return new String (str + suffix); }
    | "(" str = expression () ")"
    { /*return new String "(" + str + ")";*/
        return new String (str); /* To accept -(15) */}
    | str = unary_op () suffix = expression ()
    { return new String (str + suffix); }
    | str = network_mapping()
    { return str; }
    }

    String expression_tail () :

```

```

{
    String str = "";
    String suffix1 = "";
    String suffix2 = "";
}
{
    str = binary_op () suffix1 = expression () suffix2 =
expression_tail ()
    { return new String (str + suffix1 + suffix2); }
    | empty_string ()
    { return ""; }
}

String expression_suffix1 () :
{
    Token tok;
    String unit = "";
}
{
    "." tok = < INTEGER_LITERAL >
    { return new String ( "." + tok.toString () ); }
    | unit = unit ()
    { return unit; }
    | empty_string ()
    { return ""; }
}

String expression_suffix2 () :
{
    String str = "";
    String s = "";
}
{
    str = type_name_suffix () "." s = op_name ()
    { str = str.concat ( "." ).concat ( s ); }
    [ "(" s = expression_list () ")" { str = new String (str + "(" + s
+ ")"); } ]
    { return str; }
    | empty_string ()
    { return ""; }
}

/**
 * Production 40
 */
String op_name () :
{
    String name;
}
{
    name = id ()
    {
        return name;
    }
}

/**
 * Production 41
 */
String id () :
{

```

```

    Token tok;
}
{
    tok = < IDENTIFIER >
    {
        return tok.toString ();
    }
}

/*
    Production 42
*/
/* This is a token, it is removed from the parser for efficiency
void real_literal () :
{}
{
    < INTEGER_LITERAL > "." < INTEGER_LITERAL >
}
*/

/**
    * Production 43
    */
String integer_literal () :
{
    Token intLiteral;
}
{
    intLiteral = < INTEGER_LITERAL >
    {
        return intLiteral.toString ();
    }
}

/*
    Production 44
*/
/* This is a token, it is removed from the parser for efficiency
void string_literal () :
{}
{
    < STRING_LITERAL >
}
*/

/*
    Production 45
*/
/* This is a token, it is removed from the parser for efficiency
void digit () :
{}
{
    < DIGIT >
}
*/

/*
    Production 46
*/
/* This is a token, it is removed from the parser for efficiency
void letter () :

```

```

{}
{
    < LETTER >
}
*/

/*
    Production 47
*/
/* This goes into < IDENTIFIER >, it is removed from the parser
void alpha_numeric () :
{}
{
    letter ()
    | digit ()
    | "_"
}
*/

/*
    * Production 48
    */
/* This production goes into < TEXT > */
/*
String text () :
{
    Token text;
}
{
    text = < TEXT >
    {
        return text.toString ();
    }
}
*/

/**
    * Production 49
    */
/** Represents the empty string, not a part of the PSDL grammar */
void empty_string () :
{}
{
    {
        return;
    }
}

/*
    Production 50
*/
/* This production is no more needed
void ch_ar () :
{}
{
    < CHAR : ~["]" ] >
}
*/

/**
    * Production 50

```

```

    */
String network_mapping () :

{
    Token tok1;
    Token tok2;
}

{
    tok1 = <NETWORKMAPPING> "=" tok2 = <IDENTIFIER>
    {
        currentOp. setNetWorkLabel(tok2.toString());
        return new String( tok1.toString() + "=" + tok2.toString() );
    }
}

/**
 * This production is not in the grammar
 * It is used to check output guards of a vertex
 */
void check_output_guards () :
{
{
    ( < OUTPUT > id_list () < IF > expression () [reqmts_trace ()] )+
}

/**
 * This production is not in the grammar
 * It is used to check exception guards of a vertex
 */
void check_exception_guards () :
{
{
    ( < EXCEPTION > id () [ < IF > expression () ] [reqmts_trace ()]
)+
}

/**
 * This production is not in the grammar
 * It is used to check exception list of a vertex
 */
void check_exception_list () :
{
{
    ( < EXCEPTIONS > id_list () )+
}

/**
 * This production is not in the grammar
 * It is used to check timer ops of a vertex
 */
void check_timer_ops () :
{
{
    ( timer_op () id () [ < IF > expression () ] [reqmts_trace ()] )+
}

```

THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX D. SOURCE CODE

```

package caps;

import caps.CAPSMMain.*;

/**
 * The driver program for CAPS.
 */
public class Caps
{
    /**
     * The constructor for this class.
     *
     * @param args[] The command line parameters.
     * (No command line parameter is necessary for this program.)
     */
    public static void main (String args [])
    {
        CAPSMMainWindow main = new CAPSMMainWindow ();
    }

} // End of the class Caps
package caps;

import caps.GraphEditor.Editor;
import caps.Psdl.Vertex;
import caps.Psdl.DataTypes;
import java.io.File;

/**
 * The driver class for the PSDL Editor.
 * This class is intended to execute the Editor in a stand alone way for
 * debugging purposes.
 */
public class EditorDriver {

    /**
     * The main method for this class
     *
     * @param args The command line arguments for the driver
     */
    public static void main (String args [])
    {
        new Thread (new Editor (new File ("noname"), new File("noname"), new Vertex (0,
0, null, false), new DataTypes ())).start ();
    }

} // End of the class EditorDriver
package caps.Ada;
import java.io.StringWriter;
import caps.Psdl.*;
import java.util.*;

/**
 * Create Ada templet for CAPS machine
 * @author Shen-Yi Tao
 * @version 1.0
 */

public class AdaTemplet
{
    private static StringWriter[] writer;
    private static Vertex treeRoot;
    private String rootName;
    private String vertexName;
    private Vector inVector;
    private Vector inOutVector;
    private Vector outVector;
    private int file=0;

    /**
     * constructor
     */
    public AdaTemplet(Vertex root)

```

```

{
    int childCount = 0;
    treeRoot = root;
    rootName = treeRoot.getLabel()+"_"+treeRoot.getId();

    for(Enumeration e = treeRoot.breadthFirstEnumeration();
        e.hasMoreElements();)
    {
        DataFlowComponent DFC = (DataFlowComponent)(e.nextElement());
        if(DFC instanceof Vertex)
        {
            if(DFC.isLeaf())
                childCount++;
        }
    }
    writer = new StringWriter[childCount+2];
    inVector = new Vector();
    inOutVector = new Vector();
    outVector = new Vector();
}

/**
 * write templet to file
 */
public void writeTemplet()
{
    for(Enumeration e = treeRoot.breadthFirstEnumeration();
        e.hasMoreElements();)
    {
        DataFlowComponent DFC = (DataFlowComponent)(e.nextElement());
        String name = "";
        if(DFC instanceof Vertex )
            name = DFC.getLabel()+"_"+DFC.getId();

        if(DFC instanceof Vertex & !DFC.isRoot())
        {
            if( ((Vertex)DFC).isLeaf()
                & ((Vertex)DFC).getImpLanguage().equalsIgnoreCase("ada") )
            {
                inVector.removeAllElements();
                inOutVector.removeAllElements();
                outVector.removeAllElements();

                seperateEdge((Vertex)DFC);
                writer[file] = new StringWriter();
                writer[file].write("with ");
                writer[file].write(rootName);
                writer[file].write("_instantiations;"+ "\n");
                writer[file].write("use ");
                writer[file].write(rootName);
                writer[file].write("_instantiations;"+ "\n");
                writer[file].write("with ");
                writer[file].write(rootName);
                writer[file].write("_exceptions;"+ "\n");
                writer[file].write("use ");
                writer[file].write(rootName);
                writer[file].write("_exceptions;"+ "\n");
                //add type def
                for(Enumeration en = getTypeDef((Vertex)DFC);
                    en.hasMoreElements();)
                {
                    String st = ((String)en.nextElement() );
                    writer[file].write("with ");
                    writer[file].write(st);
                    writer[file].write("_pkg;"+ "\n");
                    writer[file].write("use ");
                    writer[file].write(st);
                    writer[file].write("_pkg;"+ "\n");
                }
                //add type def
                writer[file].write("package ");
                writer[file].write(name);
                writer[file].write("_pkg is"+ "\n");
                writer[file].write(" procedure ");
            }
        }
    }
}

```

```

        writer[file].write(name+"\n");
        writer[file].write("  ("+"\n");
        writer[file].write(getEdges());
        writer[file].write("  );"+" \n");
        writer[file].write("end ");
        writer[file].write(name);
        writer[file].write("_pkg; "+" \n");
        writer[file].write("\n");
        writer[file].write("package body ");
        writer[file].write(name);
        writer[file].write("_pkg is "+" \n");
        writer[file].write("  procedure ");
        writer[file].write(name+"\n");
        writer[file].write("  ("+"\n");
        writer[file].write(getEdges());
        writer[file].write("  ) is" + "\n");
        writer[file].write("    begin"+" \n");
        writer[file].write("      null;");
        writer[file].write("  --your implementation goes here"+" \n");
        writer[file].write("    end ");
        writer[file].write(name);
        writer[file].write("; "+" \n");
        writer[file].write("end ");
        writer[file].write(name);
        writer[file].write("_pkg; "+" \n" + "\n");
        //increase counter
        file++;
    }
}

/**
 * seperate edges into in-edge , out-edge or in-out-edge
 */
public void seperateEdge(Vertex v)
{
    Vector inEdgeName = new Vector();
    Vector outEdgeName = new Vector();
    StringWriter sw = new StringWriter();

    Vector inEdge = (Vector)( v.getInEdgesVector().clone() );
    for(Enumeration e = inEdge.elements(); e.hasMoreElements();)
    {
        Edge temp = (Edge)( e.nextElement());
        if( inEdgeName.indexOf(temp.getLabel()) == -1)
            inEdgeName.add(temp.getLabel()+"|"+temp.getStreamType());
    }
    Vector outEdge = (Vector)( v.getOutEdgesVector().clone() );
    for(Enumeration en = outEdge.elements(); en.hasMoreElements();)
    {
        Edge temp = (Edge)( en.nextElement());
        if( outEdgeName.indexOf(temp.getLabel()) == -1)
            outEdgeName.add(temp.getLabel()+"|"+temp.getStreamType());
    }

    String tempName = "";
    for(Enumeration enu = inEdgeName.elements();
        enu.hasMoreElements();)
    {
        String in = ((String)enu.nextElement());
        String inName = in;
        if(outEdgeName.indexOf(inName)==-1)
        {
            if(inVector.indexOf(inName)== -1)
                inVector.add(inName);
        }
        else
        {
            while(outEdgeName.indexOf(inName)!=-1)
            {
                outEdgeName.removeElementAt(outEdgeName.indexOf(inName));
            }
            inOutVector.add(inName);
        }
    }
}

```

```

        for(Enumeration enum = outEdgeName.elements();
            enum.hasMoreElements();)
        {
            String out = ((String)enum.nextElement());
            outVector.add( out );
        }
    }
    /**
     * get the def of all edges in the vertex
     */
    public String getEdges()
    {
        StringWriter sw = new StringWriter();
        for(Enumeration enum = inVector.elements();
            enum.hasMoreElements();)
        {
            String in = ( (String)enum.nextElement() );
            String edgeName = in.substring(0,in.indexOf("|"));
            String edgeType = in.substring(in.indexOf("|")+1);
            sw.write("      "+edgeName);
            sw.write(":");
            sw.write(" in ");
            sw.write(edgeType);
            if(enum.hasMoreElements())
                sw.write(";"+ "\n");
            else if(!outVector.isEmpty() | !inOutVector.isEmpty() )
                sw.write(";"+ "\n");
            else
                sw.write("\n");
        }
        for(Enumeration enum = outVector.elements();
            enum.hasMoreElements();)
        {
            String out = ((String)enum.nextElement());
            String edgeName = out.substring(0,out.indexOf("|"));
            String edgeType = out.substring(out.indexOf("|")+1);
            sw.write("      "+edgeName);
            sw.write(":");
            sw.write(" out ");
            sw.write(edgeType);
            if(enum.hasMoreElements())
                sw.write(";"+ "\n");
            else if(!inOutVector.isEmpty() )
                sw.write(";"+ "\n");
            else
                sw.write("\n");
        }
        for(Enumeration enum = inOutVector.elements();
            enum.hasMoreElements();)
        {
            String inOut = ((String)enum.nextElement());
            String edgeName = inOut.substring(0,inOut.indexOf("|"));
            String edgeType = inOut.substring(inOut.indexOf("|")+1);
            sw.write("      "+edgeName);
            sw.write(":");
            sw.write(" in out ");
            sw.write(edgeType);
            if(enum.hasMoreElements())
                sw.write(";"+ "\n");
            else
                sw.write("\n");
        }
        return sw.toString();
    }
}

public StringWriter [] getTemplet()
{
    return writer;
}

public boolean exist (String name ,Vector types)
{
    boolean flag = false;
    for (Enumeration enum = types.elements (); enum.hasMoreElements
        ());

```

```

        {
            if (name.equals((String) enum.nextElement ()))
                flag = true;
        }
        return flag;
    }

    public boolean isPredefined (String str)
    {
        if (str.equalsIgnoreCase ("boolean") || str.equalsIgnoreCase
            ("character")
            || str.equalsIgnoreCase ("string") || str.equalsIgnoreCase
            ("integer")
            || str.equalsIgnoreCase ("real") || str.equalsIgnoreCase
            ("exception"))
            return true;
        else
            return false;
    }

    public Enumeration getTypeDef(Vertex v)
    {
        Vector types = new Vector();
        Vector inEdge = (Vector)( v.getInEdgesVector().clone() );
        Vector outEdge = (Vector)( v.getOutEdgesVector().clone() );

        for(Enumeration e = inEdge.elements(); e.hasMoreElements(); )
        {
            Edge ed = ( (Edge)e.nextElement() );
            String temp = ed.getStreamType();

            if(! isPredefined(temp) & ! exist(temp, types) )
            {
                types.add(temp);
            }
        }

        for(Enumeration en = outEdge.elements(); en.hasMoreElements(); )
        {
            Edge ed = ( (Edge)en.nextElement() );
            String temp = ed.getStreamType();

            if( ! isPredefined(temp) & !exist(temp, types) )
            {
                types.add(temp);
            }
        }

        return types.elements();
    }

}

//end of AdaTemplet
package caps.CAPSMMain;

import javax.swing.JMenuBar;

/**
 * The menubar of the main CAPS window.
 */

/**
 * CAPS Main Window
 */

public class CAPSMMainMenuBar extends JMenuBar
{
    /**
     * The constructor for this class.
     *
     * @param owner The parent class which has declared this menubar.
     */
    public CAPSMMainMenuBar (CAPSMMainWindow owner)
    {
        super ();

        // Add the menus
    }
}

```

```

        add (new PrototypeMenu (owner));
        add (new EditMenu (owner));
        add (new DatabasesMenu ());

        add (new ExecSupportMenu (owner));
        add (new HelpMenu ());
    }

} // End of the class CAPSMainMenuBar
package caps.CAPSMMain;

import java.awt.*;
import javax.swing.*;
import java.io.File;
import caps.Builder.PsdlBuilder;
import caps.Psdl.Vertex;
import caps.Psdl.DataTypes;
import caps.GraphEditor.Editor;
import java.awt.event.*;
import java.util.Vector;
import java.util.Enumuration;
import java.io.IOException;

/**
 * The main CAPS window.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */

/**
 * CAPS main window
 */

public class CAPSMMainWindow extends JFrame
{
    /**
     * The width of the frame.
     */
    private final int WIDTH = 400;

    /**
     * The height of the frame.
     */
    private final int HEIGHT = 150;

    /**
     * The File that contains the PSDL prototype.
     */
    private File prototype;
    //add 8/26/00 SYT
    /**
     * The Folder that contains the PSDL prototype.
     */
    private File adaTemplet;

    /**
     * add private attribute to hold protoHome
     * default protoHome = $HOME in UNIX
     * and = C:\Windows in Windows
     */
    private static String protoHome;

    /**
     * add private attribute to hold protoName
     */
    private static String protoName;

    /**
     * add private attribute to hold protoVersion
     */
    private static String protoVersion;
}

```

```

    * add private attribute to hold CAPSJavaHome
    */
private static String CAPSJavaHome;

/**
 * The Vector that holds references to the open prototypes
 */
private static Vector openPrototypes;

/**
 * The constructor for this class.
 */
public CAPSMainWindow ()
{
    super ("HSI Designer Mode");    // The title of the frame.

    prototype = null;
    adaTemplet = null;

    openPrototypes = new Vector (0, 2);

    initialize ();
}

/**
 * Initializes the CAPS main window.
 */
public void initialize ()
{
    setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
    addWindowListener (new ExitCAPSMain (this));

    /**
     * Places the frame in the upper-right corner of the screen
     */
    Dimension screenSize =
        Toolkit.getDefaultToolkit().getScreenSize();

    setLocation(screenSize.width - (WIDTH + WIDTH / 2), HEIGHT /
        2);

    setResizable (false);

    setJMenuBar (new CAPSMainMenuBar (this));

    JPanel panel = new JPanel ();

    //Change the capsLabel in the following state
    // from "Heterogeneous System Integrator"
    // to "Heterogeneous Systems Integrator"

    JLabel capsLabel = new JLabel ("Heterogeneous Systems
        Integrator");
    capsLabel.setFont (new Font ("Courier", Font.BOLD, 17));

    // Add code to initialize CAPSJavaHome in order for
    // program to locate caps/Images correctly

    // The system property for the CAPS classes directory.
    CAPSJavaHome = System.getProperty ("CAPSJavaHome");
    if (CAPSJavaHome == null)
    {
        CAPSJavaHome = ".";
    }
    //System.out.println ("CAPSJavaHome = " + CAPSJavaHome);

    //JLabel imageLabel = new JLabel (new ImageIcon
    // ("caps/Images/caps.gif"));
    JLabel imageLabel =
        new JLabel (new ImageIcon (CAPSJavaHome +
            "/caps/Images/caps.gif"));

    panel.add (Box.createHorizontalStrut (5));
    panel.add (imageLabel);
    panel.add (Box.createHorizontalStrut (5));
    panel.add (capsLabel);
    panel.add (Box.createHorizontalStrut (5));

```

```

        getContentPane ().add (panel);

        pack ();

        setVisible (true);
    }

    /**
     * Sets the prototype file to the argument.
     *
     * @param f The File that contains the PSDL prototype.
     */
    public void setPrototype (File f)
    {
        prototype = f;
        //added the following debug statement
        //System.out.println ("Prototype Name = " + prototype.getName());
        //System.out.println ("Prototype Name Length = "
        //                    + (prototype.getName()).length());
    }

    //add 8/26/00 SYT
    /**
     * Sets the add templet file to the argument.
     *
     * @param t The File to create the ada templet.
     */
    public void setAdaTemplet(File t)
    {
        adaTemplet = t;
    }

    /**
     * Sets the prototype home directory name to the argument.
     *
     * @param s The string that contains the prototype home dir.
     */
    public void setProtoHome (String s)
    {
        protoHome = s;
        // debug statement
        System.out.println ("Prototype Home = " + s);
    }

    /**
     * Sets the prototype name to the argument.
     *
     * @param s The string that contains the prototype name.
     */
    public void setProtoName (String s)
    {
        protoName = s;
        // debug statement
        System.out.println ("Prototype Name = " + s);
    }

    /**
     * Sets the prototype name to the argument.
     *
     * @param s The string that contains the prototype name.
     */
    public void setProtoVersion (String s)
    {
        protoVersion = s;
        // debug statement
        System.out.println ("Prototype Version = " + s);
    }

    /**
     * Returns the vector that holds the open prototype files.
     *
     * @return the vector that contains the open prototype files.
     */
    public Vector getOpenPrototypes ()
    {
        return openPrototypes;
    }

```



```

}

/**
 * Opens the graphics editor to edit a prototype.
 * Translate file to symbols      SYT
 */
public void editPrototype ()
{
    if (prototype == null)
    {
        // No prototype is selected to open
        JOptionPane.showMessageDialog (this,"No prototype is selected
            to edit."
            , "Error Message", JOptionPane.ERROR_MESSAGE);
    }
    else if (!isPrototypeChanged ())
    {
        // Attempt to edit the same prototype.
        JOptionPane.showMessageDialog (this
            , new String ("Prototype " + prototype.getName () + " is
                already open.")
            , "Error Message", JOptionPane.ERROR_MESSAGE);
    }

    else
    {
        //read and translate prototype from text to graphics. SYT
        PsdlBuilder.disable_tracing ();          // Disable debug
                                                //messages

        Vertex root = null;
        root = PsdlBuilder.buildPrototype (prototype);
        if (root == null)
        {
            // If this is a new prototype ,Prototype name
            // is the same as the file name. SYT
            root = new Vertex (0, 0, null, false);
            String name = prototype.getName ();
            root.setLabel (name.substring (0, name.length () - 5));
        }
        DataTypes types = new DataTypes ();
        types.buildTypes (prototype);

        Editor e = new Editor (prototype, adaTemplet, root, types);
        new Thread (e).start ();
        openPrototypes.addElement (e);
    }
}

/**
 * Checks whether or not the current prototype file is already used
 * by a PSDL Editor.
 *
 * @return true if one of the open prototypes is the same as the
 *
 * current prototype file.
 */
public boolean isPrototypeChanged ()
{
    for (Enumeration enum = openPrototypes.elements ()
        ; enum.hasMoreElements ();)
    {
        Editor e = (Editor) enum.nextElement ();
        if (prototype.equals (e.getPrototypeFile ()))
            return false;
    }
    return true;
}

/**
 * Removes one element from the openPrototypes vector.
 *
 * @param e the editor that is going to be removed from the vector.
 */
public static void removeEditor (Editor e)
{
    openPrototypes.removeElement (e);
}

/**
 * Checks if the status of any of the open prototypes is

```

```

    * 'saveRequired'.
    * Prompts the user to save the prototype.
    *
    * @return true if none of the prototypes need saving.
    */
public boolean isOpenPrototypeSaved ()
{
    boolean flag = true;
    Editor e;
    label :
    for (Enumeration enum = openPrototypes.elements ()
        ;enum.hasMoreElements ();)
    {
        e = (Editor) enum.nextElement ();
        if (e.isSaveRequired ())
        {
            int ix = JOptionPane.showConfirmDialog
                (this, new String ("Save changes to the
                    prototype " +
                        e.getRoot ().getLabel () +
                        "?"));
            if (ix == JOptionPane.CANCEL_OPTION)
            {
                flag = false;
                break label;
            }
            else if (ix == JOptionPane.YES_OPTION)
                e.savePrototype ();
        }
    }
    return flag;
}

// added procedure translatePrototype()
/**
 * printing run time project info.
 */
public void translatePrototype()
{
    String command = "translate.script " + protoHome + " " +
        protoName + " "
        + protoVersion;
    System.out.println (command);
    try
    {
        Runtime run = Runtime.getRuntime ();
        run.exec (command);
    } catch (IOException ex)
    {
        System.out.println (ex);
    }
}

// added procedure schedulePrototype()
public void schedulePrototype()
{
    String command = "make.script " + protoHome + " " + protoName
        + " "
        + protoVersion;
    System.out.println (command);
    try
    {
        Runtime run = Runtime.getRuntime ();
        run.exec (command);
    } catch (IOException ex)
    {
        System.out.println (ex);
    }
}

//added procedure schedulePrototype()
public void compilePrototype()
{
    String command = "compile.script " + protoHome + " " +
        protoName + " "
        + protoVersion;
    System.out.println (command);
}

```

```

        try
        {
            Runtime run = Runtime.getRuntime ();
            run.exec (command);
        }
        catch (IOException ex)
        {
            System.out.println (ex);
        }
    }

    //added procedure schedulePrototype()
    public void executePrototype()
    {
        String command = "execute.script " + protoHome + " " +
            protoName + " "
            + protoVersion;
        System.out.println (command);
        try
        {
            Runtime run = Runtime.getRuntime ();
            run.exec (command);
        }
        catch (IOException ex)
        {
            System.out.println (ex);
        }
    }

} // End of the class CAPSMainWindow
package caps.CAPSMain;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class holds the 'Databases' menu items.
 *
 * @author SYT
 * @version 1.1
 */
public class DatabasesMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'Design Database' event
     */
    private JMenuItem designDBMenuItem = new JMenuItem ("Design
        Database");

    /**
     * Initiates the 'Software Base' event
     */
    private JMenuItem swBaseMenuItem = new JMenuItem ("Software Base");

    /**
     * Constructor for this class.
     */
    public DatabasesMenu ()
    {
        super ("Databases");

        add (designDBMenuItem);
        add (swBaseMenuItem);

        // Register the action listeners SYT
        designDBMenuItem.addActionListener (this);
        swBaseMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     * @param e The action event that is created by selecting
     * a menu item from this menu

```

```

        */
        public void actionPerformed(ActionEvent e)
        {
            if (e.getSource () == designDBMenuItem)
            {
                System.out.println ("Design DB has not been implemented yet");
            }
            else if (e.getSource () == swBaseMenuItem)
            {
                System.out.println ("SW Base has not been implemented yet");
            }
        }
    }

} // End of the class DatabasesMenu
package caps.CAPSMain;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class holds the 'Edit' menu items.
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class EditMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'PSDL' event
     */
    private JMenuItem psdlMenuItem = new JMenuItem ("PSDL");

    /**
     * Initiates the 'Ada' event
     */
    private JMenuItem adaMenuItem = new JMenuItem ("Ada");

    /**
     * Initiates the 'Interface' event
     */
    private JMenuItem interfaceMenuItem = new JMenuItem ("Interface");

    /**
     * Initiates the 'Requirements' event
     */
    private JMenuItem requirementsMenuItem = new JMenuItem ("Requirements");

    /**
     * Initiates the 'Change Request' event
     */
    private JMenuItem changeReqMenuItem = new JMenuItem ("Change Request");

    /**
     * Initiates the 'CAPS Defaults' event
     */
    private JMenuItem capsDefaultsMenuItem = new JMenuItem ("HSI Defaults");

    /**
     * Initiates the 'Hardware Model' event
     */
    private JMenuItem hwModelMenuItem = new JMenuItem ("Hardware Model");

    /**
     * The main window which owns this menu.
     */
    protected CAPSMainWindow owner;

    /**
     * The constructor for this class.
     *
     * @param f The parent class which has declared this menubar.
     */
    public EditMenu (CAPSMainWindow f)
    {
        super ("Edit");
    }

```

```

        owner = f;

        add (psdlMenuItem);
        add (adaMenuItem);
        add (interfaceMenuItem);
        add (requirementsMenuItem);
        add (changeReqMenuItem);
        add (capsDefaultsMenuItem);
        add (hwModelMenuItem);

        /*
         * Register the action listeners
         */
        psdlMenuItem.addActionListener (this);
        adaMenuItem.addActionListener (this);
        interfaceMenuItem.addActionListener (this);
        requirementsMenuItem.addActionListener (this);
        changeReqMenuItem.addActionListener (this);
        capsDefaultsMenuItem.addActionListener (this);
        hwModelMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting a menu
     * item from this menu
     */
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == psdlMenuItem)
        {
            //create graphics.   SYT
            owner.editPrototype ();
        }
        else if (e.getSource () == adaMenuItem)
        {
            System.out.println ("Ada Editor has not been implemented
                                yet");
        }
        else if (e.getSource () == interfaceMenuItem)
        {
            System.out.println ("Interface Editor has not been implemented
                                yet");
        }
        else if (e.getSource () == requirementsMenuItem)
        {
            System.out.println ("Requirements Editor has not been
                                implemented yet");
        }
        else if (e.getSource () == changeReqMenuItem)
        {
            System.out.println ("Change Requirements has not been
                                implemented yet");
        }
        else if (e.getSource () == capsDefaultsMenuItem)
        {
            System.out.println ("CAPS Defaults has not been implemented
                                yet");
        }
        else if (e.getSource () == hwModelMenuItem)
        {
            System.out.println ("Hardware Model has not been implemented
                                yet");
        }
    }
}

} // End of the class EditMenu
package caps.CAPSMain;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

```

```

/**
 * This class holds the 'Exec Support' menu items.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
/**
 * Changes:
 *   added the owner parameter to the ExecSupportMenu constructor
 *   added the protected attribute ownerWindow to the ExecSupportMenu class
 *   added the call to ownerWindow.translatePrototype() to actionPerformed method
 *   added the call to ownerWindow.schedulePrototype(),
 *   ownerWindow.compilePrototype() and ownerWindow.executePrototype()
 *   to actionPerformed method
 */

public class ExecSupportMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'Translate' event
     */
    private JMenuItem translateMenuItem = new JMenuItem ("Translate");

    /**
     * Initiates the 'Schedule' event
     */
    private JMenuItem scheduleMenuItem = new JMenuItem ("Schedule");

    /**
     * Initiates the 'Compile' event
     */
    private JMenuItem compileMenuItem = new JMenuItem ("Compile");

    /**
     * Initiates the 'Execute' event
     */
    private JMenuItem executeMenuItem = new JMenuItem ("Execute");

    /**
     * added the ownerWindow attribute
     *
     * The main window which owns this menu.
     */
    protected CAPSMainWindow ownerWindow;

    /**
     * Constructor for this class.
     */

    // added the owner parameter to the constructor

    public ExecSupportMenu (CAPSMainWindow owner)
    {
        super ("Exec Support");

        // added the statement to update the ownerWindow attribute
        ownerWindow = owner;

        add (translateMenuItem);
        add (scheduleMenuItem);
        add (compileMenuItem);
        add (executeMenuItem);

        /*
         * Register the action listeners
         */
        translateMenuItem.addActionListener (this);
        scheduleMenuItem.addActionListener (this);
        compileMenuItem.addActionListener (this);
        executeMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     * @param e The action event that is created by selecting a menu
     * item from this menu

```

```

    */
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == translateMenuItem)
        {
            ownerWindow.translatePrototype();
        }
        else if (e.getSource () == scheduleMenuItem)
        {
            ownerWindow.schedulePrototype();
        }
        else if (e.getSource () == compileMenuItem)
        {
            ownerWindow.compilePrototype();
        }
        else if (e.getSource () == executeMenuItem)
        {
            //System.out.println ("Executing telnet");
            //try {
            //    Runtime run = Runtime.getRuntime ();
            //    run.exec ("telnet.exe");
            //} catch (IOException ex) {
            //    System.out.println (ex);
            //}
            ownerWindow.executePrototype();
        }
    }

} // End of the class ExecSupportMenu
package caps.CAPSMMain;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/**
 * Closes the caps main window and exits from the program.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
class ExitCAPSMMain extends WindowAdapter
{
    /**
     * The main program that has declared this object
     */
    CAPSMMainWindow capsMain;

    /**
     * The constructor for this class.
     *
     * @param owner The parent class which has declared this menubar.
     */
    public ExitCAPSMMain (CAPSMMainWindow caps)
    {
        capsMain = caps;
    }

    /**
     * Window event handler for the menu events.
     *
     * @param e The window event that is created when the program close
     * icon is pressed.
     */
    public void windowClosing(WindowEvent e)
    {
        // Exit the program if the prototypes are saved
        if (capsMain.isOpenPrototypeSaved ())
            System.exit (0);
    }

} // End of the class ExitCapsMain
package caps.CAPSMMain;

import javax.swing.JMenu;

```

```

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/**
 * This class implements the 'Help' menu.
 */
public class HelpMenu extends JMenu implements ActionListener
{
    /**
     * Constructor for this class.
     */
    public HelpMenu ()
    {
        super ("Help");
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting
     * a menu item from this menu
     */
    public void actionPerformed(ActionEvent e)
    {
        // Not implemented yet
    }

} // End of the class HelpMenu
package caps.CAPSMMain;

import javax.swing.*;
import javax.swing.filechooser.FileSystemView;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.Vector;

/**
 * This class holds the 'Prototype' menu items.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */

/**
 * Changes:
 *
 * add statement to set protoHome, protoName and protoVersion
 * of CAPSMMainWindow object in processNewMenuItem and
 * processOpenMenuItem
 *
 * change code in processNewMenuItem and processOpenMenuItem
 * so that user only have to specify the $HOME directory
 * instead of $HOME/.caps using the -DPROTOTYPEHOME flag
 */

public class PrototypeMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'New' event
     */
    private JMenuItem newMenuItem = new JMenuItem ("New");

    /**
     * Initiates the 'Open' event
     */
    private JMenuItem openMenuItem = new JMenuItem ("Open");

    /**
     * Initiates the 'Commit Work' event
     */
    private JMenuItem commitWorkMenuItem = new JMenuItem ("Commit
        Work");

    /**
     * Initiates the 'Retrieve From DDB' event

```



```

    */
    private JMenuItem retrieveMenuItem = new JMenuItem ("Retrieve From
        DDB");

    /**
     * Initiates the 'Quit' event
     */
    private JMenuItem quitMenuItem = new JMenuItem ("Quit");

    /**
     * The main window which owns this menu.
     */
    protected CAPSMainWindow ownerWindow;

    /**
     * Constructor for this class.
     *
     * @param owner The main window which has created this menu.
     */
    public PrototypeMenu (CAPSMainWindow owner)
    {
        super ("Prototype");

        ownerWindow = owner;

        add (newMenuItem);
        add (openMenuItem);
        add (commitWorkMenuItem);
        add (retrieveMenuItem);
        add (quitMenuItem);

        /*
         * Register the action listeners
         */
        newMenuItem.addActionListener (this);
        openMenuItem.addActionListener (this);
        commitWorkMenuItem.addActionListener (this);
        retrieveMenuItem.addActionListener (this);
        quitMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting
     * a menu item from this menu
     */
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == newMenuItem)
        {
            processNewMenuItem ();
        }
        else if (e.getSource () == openMenuItem)
        {
            processOpenMenuItem ();
        }
        else if (e.getSource () == commitWorkMenuItem)
        {
            System.out.println ("Commit Work has not yet been
                implemented");
        }
        else if (e.getSource () == retrieveMenuItem)
        {
            System.out.println ("Retrieve has not yet been implemented");
        }
        else if (e.getSource () == quitMenuItem)
        {
            // Exit the program if all of the prototypes are saved.
            if (ownerWindow.isOpenPrototypeSaved ())
                System.exit (0);
        }
    }

    /**
     * Handles the event which is caused by selecting the 'New' menu
     * item.

```

```

*/
public void processNewMenuItem ()
{
    // The system property for the home prototype directory.
    String protoHome = System.getProperty ("PROTOTYPEHOME");
    File protoDir;
    // add local variable proto and version
    String proto;
    String version;

    if (protoHome == null)
    {
        // If it is not set as a command line argument
        File homeDir = FileSystemView.getFileSystemView
            ().getHomeDirectory ();
        //protoHome = new String (homeDir + File.separator + ".caps");
        //protoDir = new File (protoHome);
        protoHome = homeDir.toString();
        protoDir = new File (protoHome + File.separator + ".caps");
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }
    else
    {
        // add the arguments File.separator + ".caps" to
        // the following statement since protoHome now only
        // contains $HOME instead of $HOME/.caps.
        protoDir = new File (protoHome + File.separator + ".caps");
        //protoDir = new File (protoHome);
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }

    // moved String proto declaration to beginning of method
    // String proto = JOptionPane.showInputDialog (ownerWindow,

    proto = JOptionPane.showInputDialog (ownerWindow,
        "Enter Prototype Name : ", "New",
        JOptionPane.PLAIN_MESSAGE);
    if (proto == null)
        return;

    // moved String version declaration to beginning of method
    //String version = JOptionPane.showInputDialog (ownerWindow,

    version = JOptionPane.showInputDialog (ownerWindow,
        "Prototype Version Information : ", "New"
        , JOptionPane.PLAIN_MESSAGE);
    {
        String name = proto;

        // 7-12-99 add the arguments File.separator + ".caps" to
        // the following statement since protoHome now only
        // contains $HOME instead of $HOME/.caps.
        File file = new File (protoHome + File.separator + ".caps"
            + File.separator + proto +
            File.separator
            + version + File.separator + name
            + ".psdl");
        //add 8/26/00 SYT
        File adaTemplet = new File (protoHome + File.separator +
            ".caps"
            + File.separator + proto +
            File.separator
            + version + File.separator );

        if (file.exists ())
        {
            int selected = JOptionPane.showConfirmDialog (ownerWindow
                , "Selected prototype file already
                exists.\n"
                + "Do you want to overwrite it ?");
            if (selected == JOptionPane.YES_OPTION)
            {
                try
                {
                    file.delete ();
                    file.createNewFile ();
                }
            }
        }
    }
}

```

```

        adaTemplet.delete();
        adaTemplet.createNewFile();
    }
    catch (java.io.IOException ex)
    {
        System.out.println (ex);
    }
    ownerWindow.setPrototype (file);
    //add 8/26/00 SYT
    ownerWindow.setAdaTemplet(adaTemplet);

    // add statement to invoke setProtoHome, setProtoName,
    // and setProtoVersion
    ownerWindow.setProtoHome(protoHome);
    ownerWindow.setProtoName(proto);
    ownerWindow.setProtoVersion(version);
}
}
else {
    try {
        File dir = file.getParentFile ().getParentFile ();
        dir.mkdir ();
        File vers = file.getParentFile ();
        vers.mkdir ();
        file.createNewFile ();
        adaTemplet.createNewFile();

    }
    catch (java.io.IOException ex)
    {
        System.out.println (ex);
    }
    ownerWindow.setPrototype (file);
    //add 8/26/00 SYT
    ownerWindow.setAdaTemplet(adaTemplet);
    // add statement to invoke setProtoHome, setProtoName,
    // and setProtoVersion
    ownerWindow.setProtoHome(protoHome);
    ownerWindow.setProtoName(proto);
    ownerWindow.setProtoVersion(version);
}
}
}

/**
 * Handles the event which is caused by selecting the 'Open' menu *
 * item.
 */
public void processOpenMenuItem ()
{
    String protoHome = System.getProperty ("PROTOTYPEHOME");
    File protoDir;
    if (protoHome == null)
    {
        // If it is not set as a command line argument
        File homeDir = FileSystemView.getFileSystemView
            ().getHomeDirectory ();
        //protoHome = new String (homeDir + File.separator + ".caps");
        //protoDir = new File (protoHome);
        protoHome = homeDir.toString();
        protoDir = new File (protoHome + File.separator + ".caps" );
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }
    else
    {
        // 7-12-99 add the arguments File.separator + ".caps" to
        // the following statement since protoHome now only
        // contains $HOME instead of $HOME/.caps.
        protoDir = new File (protoHome + File.separator + ".caps");
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }

    Vector prototypeNames = new Vector (0, 2);
    File [] dirs = protoDir.listFiles ();
    String protoName = "";

```

```

if (dirs.length == 0)
{
    JOptionPane.showMessageDialog
        (ownerWindow, "No prototype is found to open",
            "Error Message",
            JOptionPane.ERROR_MESSAGE);
}
else
{
    for (int ix = 0; ix < dirs.length; ix++)
    {
        protoName = dirs [ix].getName ();
        File subDirs [] = dirs [ix].listFiles ();
        for (int jx = 0; jx < subDirs.length; jx++)
        {
            prototypeNames.addElement (protoName.concat
                (File.separator + subDirs [jx].getName ()));
        }
    }

    Object [] protos = prototypeNames.toArray ();
    String selected = (String) JOptionPane
        .showInputDialog (ownerWindow
            , "Select a prototype : ", "Open"
            , JOptionPane.INFORMATION_MESSAGE, null
            , protos, protos [0]);

    if (selected != null)
    {
        // 7-12-99 add the arguments File.separator + ".caps" to
        // the following statement since protoHome now only
        // contains $HOME instead of $HOME/.caps.
        File selectedDir = new File (protoHome + File.separator +
            ".caps"
            + File.separator + selected);
        File file = new File (selectedDir.getAbsolutePath ()
            + File.separator + selectedDir
            .getParentFile ().getName () + ".psdl");
        //add 8/26/00 SYT
        /*File adaPath = new File (protoHome + protoHome + File.separator
            + ".caps" + File.separator + proto
            + File.separator
            + version + File.separator);
        */
        File adaTemplet = new File (selectedDir.getAbsolutePath () );

        // debug statements
        //System.out.println ("Prototype Home = " + protoHome);
        //System.out.println ("Directory name = " + selectedDir.toString());
        //System.out.println ("Prototype name = " + file.toString());

        if (!file.exists ())
            JOptionPane.showMessageDialog (ownerWindow
                , "The selected prototype file cannot be
                opened"
                , "Error Message"
                , JOptionPane.ERROR_MESSAGE);
        ownerWindow.setPrototype (file);
        //add 8/26/00 SYT
        ownerWindow.setAdaTemplet(adaTemplet);
        // add statement to invoke setProtoHome, setProtoName,
        // and setProtoVersion
        ownerWindow.setProtoHome(protoHome);
        ownerWindow.setProtoName(selectedDir.getParentFile().getName());
        ownerWindow.setProtoVersion(selectedDir.getName());
    }
}
} // End of the class PrototypeMenu
package caps.Display;

import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;

```

```

import javax.swing.JLabel;
import caps.Psdl.DataFlowComponent;
import java.util.Vector;

/**
 * This is an abstract super class of EdgePath and DisplayVertex.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
public abstract class DisplayComponent
{
    /**
     * The size of the Handles.
     */
    public static final int HANDLESIZE = 6;

    /**
     * The DataFlowComponent that this object associates with.
     */
    protected DataFlowComponent dfc;

    /**
     * The shape of the label of the component.
     */
    TextLayout labelShape;

    /**
     * The shape of the met of the component .
     */
    TextLayout metShape;

    /**
     * The constructor is protected so it cannot be instantiated
     * directly.
     *
     * param d the DataFlowComponent that is associated with this o
     * object.
     */
    protected DisplayComponent (DataFlowComponent d)
    {
        dfc = d;
        labelShape = null;
        metShape = null;
    }

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract Shape getShape ();

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract boolean containsClickedPoint (int xLoc, int yLoc);

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract Vector getHandles ();

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract void update ();

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract void delete ();

    /**
     * Gets the label from the DataFlowComponent and creates a
     * TextLayout shape for the label.
     *
     * @param g2D the graphics context of the DrawPanel

```

```

    */
    public void setLabelShape (Graphics2D g2D)
    {
        labelShape = new TextLayout (dfc.getLabel (), dfc.getFont (),
        g2D.getFontRenderContext ());
    }

    /**
     * Returns the bounding rectangle of the label shape.
     *
     * @return the bounding rectangle of the label shape.
     */
    public Rectangle2D getLabelShapeBounds ()
    {
        Rectangle2D r2D = labelShape.getBounds ();
        int x = dfc.getX () + dfc.getLabelXOffset () - (int) labelShape.getBounds
        ().getWidth () / 2;
        int y = dfc.getY () + dfc.getLabelYOffset () - (int)
        labelShape.getBounds ().getHeight () / 2;
        r2D.setRect (x, y, r2D.getWidth (), r2D.getHeight ());
        return r2D;
    }

    /**
     * draw green color label 2/21/00 SYT
     * Gets the location of the label shape and draws it into the
     * DrawPanel.
     * @param g2D the graphics context of the DrawPanel.
     */
    public void drawGreenLabelShape (Graphics2D g2D)
    {
        g2D.setColor(Color.green);
        int x = dfc.getX () + dfc.getLabelXOffset () - (int)
        labelShape.getBounds ().getWidth () / 2;
        int y = dfc.getY () + dfc.getLabelYOffset () + (int)
        labelShape.getBounds ().getHeight () / 2;
        labelShape.draw (g2D, x, y);
        g2D.setColor(Color.black);
    }

    /**
     * Gets the location of the label shape and draws it into the
     * DrawPanel.
     *
     * @param g2D the graphics context of the DrawPanel.
     */
    public void drawLabelShape (Graphics2D g2D)
    {
        int x = dfc.getX () + dfc.getLabelXOffset () - (int)
        labelShape.getBounds ().getWidth () / 2;
        int y = dfc.getY () + dfc.getLabelYOffset () + (int)
        labelShape.getBounds ().getHeight () / 2;
        labelShape.draw (g2D, x, y);
    }

    /**
     * Creates a vector that holds the handles of a string (met or
     * label).
     *
     * @param r2D the bounding rectangle of the string.
     * @return returns the Vector that holds the handles.
     */
    public Vector getStringHandles (Rectangle2D r2D)
    {
        Vector v = new Vector ();
        int i = HANDLESIZE / 2;
        v.add (new Rectangle2D.Double (r2D.getMinX () - i
        , r2D.getMinY () - i, HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (r2D.getMaxX () - i
        , r2D.getMinY () - i, HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (r2D.getMinX () - i
        , r2D.getMaxY () - i, HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (r2D.getMaxX () - i
        , r2D.getMaxY () - i, HANDLESIZE, HANDLESIZE));
        return v;
    }

```

```

    }

    /**
     * Gets the met (or latency) from the DataFlowComponent
     * and creates a TextLayout shape for the met.
     *
     * @param g2D the graphics context of the DrawPanel
     */
    public void setMetShape (Graphics2D g2D)
    {
        if (dfc.getMet () != null)    // It may not have an met
            metShape = new TextLayout (dfc.getMet ().toString ()
                                     , dfc.getMet1Font (), g2D.getFontRenderContext ());
        else
            metShape = new TextLayout (" ", dfc.getMet1Font ()
                                     , g2D.getFontRenderContext ());
    }

    /**
     * Returns the bounding rectangle of the met (or latency) shape.
     *
     * @return the bounding rectangle of the met (or latency) shape.
     */
    public Rectangle2D getMetShapeBounds ()
    {
        Rectangle2D r2D = metShape.getBounds ();
        int x = dfc.getX () + dfc.getMetXOffset ()
              - (int) metShape.getBounds ().getWidth () / 2;
        int y = dfc.getY () + dfc.getMetYOffset ()
              - (int) metShape.getBounds ().getHeight () / 2;
        r2D.setRect (x, y, r2D.getWidth (), r2D.getHeight ());
        return r2D;
    }

    /**
     * Gets the location of the met (or latency) shape and draws it
     * into the DrawPanel.
     *
     * @param g2D the graphics context of the DrawPanel.
     */
    public void drawMetShape (Graphics2D g2D)
    {
        if (dfc.getMet () != null)
        {
            int x = dfc.getX () + dfc.getMetXOffset ()
                  - (int) metShape.getBounds ().getWidth () / 2;
            int y = dfc.getY () + dfc.getMetYOffset ()
                  + (int) metShape.getBounds ().getHeight () / 2;
            metShape.draw (g2D, x, y);
        }
    }

    /**
     * Returns the DataFlowComponent that is associated with this
     * object.
     *
     * @return the DataFlowComponent that is associated with this
     * object.
     */
    public DataFlowComponent getDataFlowComponent ()
    {
        return dfc;
    }
}

// End of the class DisplayComponent
package caps.Display;

import java.awt.geom.*;
import java.awt.*;
import java.awt.font.*;
import caps.Psdl.*;
import java.util.Vector;

/**
 * An instance of this class is created when external streams are
 * created.
 */

```

```

public class DisplayExternal extends DisplayComponent
{
    /**
     * The External object that is associated with this object.
     */
    protected External external;

    /**
     * The shape of the External.
     */
    protected Rectangle2D.Double shape;

    /**
     * The constructor for this class.
     *
     * @param e the External that is associated with this object.
     */
    public DisplayExternal (External e)
    {
        super (e);
        external = e;
        shape = new Rectangle2D.Double (e.getX (), e.getY (), 0.5, 0.5);
    }

    /**
     * Sets the location of this shape on the DrawPanel
     */
    public void setLocation ()
    {
        double x = external.getX ();
        double y = external.getY ();
        shape setFrame (x, y, shape.getWidth (), shape.getHeight ());
    }

    /**
     * Updates the location and the width of this shape.
     */
    public void update ()
    {
        setLocation ();
    }

    /**
     * Always returns false since the shape is not displayed in the
     * DrawPanel.
     *
     * @param xLoc the x location of the clicked point.
     * @param yLoc the y location of the clicked point.
     * @return false.
     */
    public boolean containsClickedPoint (int xLoc, int yLoc)
    {
        return false;
    }

    /**
     * Returns the vector that contains the handles of the shape.
     *
     * @return an empty Vector.
     */
    public Vector getHandles ()
    {
        Vector v = new Vector ();
        return v;
    }

    /**
     * Returns the shape that represents the External.
     *
     * @return the shape that represents the External.
     */
    public Shape getShape ()
    {
        return (Shape) shape;
    }
}

```



```

        * Deletes the external that is associated with this object.
        */
    public void delete ()
    {
        external.delete ();
        external = null;
        shape = null;
    }

} // End of the class DisplayExternal
package caps.Display;

import java.awt.geom.*;
import java.awt.*;
import java.awt.font.*;
import caps.Psdl.*;
import java.util.Vector;

/**
 * This class holds a shape for its associated Vertex.
 * It can either be a rectangle for terminators or it can be a circle
 * for the operators.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class DisplayVertex extends DisplayComponent
{
    /**
     * The Vertex that is associated with this object.
     */
    protected Vertex vertex;

    /**
     * The shape of the Vertex.
     */
    protected RectangularShape shape;

    /**
     * The constructor for this class.
     *
     * param v the Vertex that is associated with this object.
     */
    public DisplayVertex (Vertex v)
    {
        super (v);
        vertex = v;
        setShape ();
        setWidth ();
        setLocation ();
    }
    //add 1/16/00 SYT
    /**
     * get current display vertex
     */
    public Vertex getVertex()
    {
        return vertex;
    }

    /**
     * Sets the location of this shape on the DrawPanel
     */
    public void setLocation ()
    {
        // x and y represent the upper left corner of the shape
        double x = vertex.getX () - shape.getWidth () / 2;
        double y = vertex.getY () - shape.getHeight () / 2;
        shape setFrame (x, y, shape.getWidth (), shape.getHeight ());
    }

    /**
     * Sets the width of this shape.
     */
    public void setWidth ()
    {

```

```

        double width = vertex.getWidth ();
        double height = vertex.getHeight ();
        shape.setFrame (vertex.getX () - shape.getWidth () / 2,
            vertex.getY ()
                - shape.getHeight () / 2, width, height);
    }

    /**
     * Updates the location and the width of this shape.
     */
    public void update ()
    {
        setLocation ();
        setWidth ();
    }

    /**
     * Checks whether the bounding box of the shape contains
     * the the location where the mouse is clicked.
     *
     * @param xLoc the x location of the clicked point.
     * @param yLoc the y location of the clicked point.
     * @return true if the bounding box contains the clicked point.
     */
    public boolean containsClickedPoint (int xLoc, int yLoc)
    {
        return getShape ().contains (new Point (xLoc, yLoc));
    }

    /**
     * Returns the vector that contains the handles of the shape.
     *
     * @return the vector that contains the handles of the shape
     */
    public Vector getHandles ()
    {
        Vector v = new Vector ();
        RectangularShape s = (RectangularShape) getShape ();
        int i = HANDLESIZE / 2;
        v.add (new Rectangle2D.Double (s.getMinX () - i
            , s.getMinY () - i, HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (s.getMaxX () - i
            , s.getMinY () - i, HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (s.getMinX () - i
            , s.getMaxY () - i, HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (s.getMaxX () - i
            , s.getMaxY () - i, HANDLESIZE, HANDLESIZE));
        return v;
    }

    /**
     * Sets the shape of this object to a circle if the
     * associated Vertex is an operator or sets it to a
     * rectangle if the Vertex is a Terminator SYT
     */
    public void setShape ()
    {
        if (vertex.isTerminator ())
            shape = new Rectangle2D.Double ();
        else
            shape = new Ellipse2D.Double ();
    }

    /**
     * Returns the shape that represents the Vertex.
     *
     * @return the shape that represents the Vertex.
     */
    public Shape getShape ()
    {
        return (Shape) shape;
    }

    /**
     * This method is called if the Vertex is composite.
     * It calculates and returns a smaller inner shape.
     */

```

```

        * @return the inner shape for the composite Vertex.
        */
public Shape getInnerShape ()
{
    if (vertex.isTerminator ())
        return (Shape) new Rectangle2D.Double (shape.getX () + 4,
            shape.getY () + 4,
            shape.getWidth () - 8,
            shape.getHeight () - 8);
    else
        return (Shape) new Ellipse2D.Double (shape.getX () + 4,
            shape.getY () + 4,
            shape.getWidth () - 8,
            shape.getHeight () - 8);
}

/**
 * Returns a shape that is slightly smaller than the shape of this
 * object.
 * The shape that is returned will be painted with the color of the
 * Vertex.
 * @return a shape that is slightly smaller than the shape of th
 * object.
 */
public Shape getPaintedShape ()
{
    if (vertex.isTerminator ())
    {
        if (vertex.isLeaf ())
            return (Shape) new Rectangle2D.Double
                (shape.getX () + 1, shape.getY () + 1
                , shape.getWidth () - 1.0f, shape.getHeight () -
                1.0f);
        else
            return (Shape) new Rectangle2D.Double
                (shape.getX () + 5, shape.getY ()
                + 5, shape.getWidth () - 10, shape.getHeight () -
                10);
    }
    else
    {
        if (vertex.isLeaf ())
            return (Shape) new Ellipse2D.Double
                (shape.getX () + 1f , shape.getY () + 1f,
                shape.getWidth ()
                - 2f, shape.getHeight () - 2f);
        else
            return (Shape) new Ellipse2D.Double
                (shape.getX () + 5, shape.getY () + 5,
                shape.getWidth ()
                - 10, shape.getHeight () - 10);
    }
}

/**
 * Deletes the vertex that is associated with this object.
 */
public void delete ()
{
    vertex.delete ();
    vertex = null;
    shape = null;
}

} // End of the class DisplayVertex
package caps.Display;

import caps.Psdl.Edge;
import java.util.*;
import java.awt.*;
import java.awt.geom.*;

/**
 * This class represents an Edge on the DrawPanel.
 * It contains a GeneralPath shape to represent the Edge.

```

```

*
* @author Shen-Yi Tao
* @version 1.1
*/
public class EdgePath extends DisplayComponent
{
    /**
     * The Edge that is associated with this object.
     */
    protected Edge edge;

    /**
     * The shape of the Edge.
     */
    protected GeneralPath shape;

    /**
     * The constructor for this class.
     *
     * param e the Edge that is associated with this object.
     */
    public EdgePath (Edge e)
    {
        super (e);
        edge = e;
        shape = new GeneralPath ();
    }

    /**
     * Returns the shape that represents the Edge.
     *
     * @return the shape that represents the Edge.
     */
    public Shape getShape ()
    {
        return (Shape) shape;
    }
    //add 2/3/00 SYT
    /**
     * draw segment between curent point to the new point
     */
    public void quadTo(float x2,float y2 )
    {
        if( shape.getCurrentPoint() != null)
        {
            Point2D temp = shape.getCurrentPoint();
            shape.quadTo(((float)temp.getX()),((float)temp.getY()),x2,y2);
        }
        shape.moveTo(x2,y2);
    }

    //add 2/3/00 SYT
    /**
     * reset to empty
     */
    public void reset()
    {
        shape.reset();
    }

    //add SYT 1/16/00 SYT
    /**
     * get this representing edge
     */
    public Edge getEdge()
    {
        return edge;
    }

    /**
     * Checks whether the bounding box of the shape contains the the
     * location
     * where the mouse is clicked.
     *
     * @param xLoc the x location of the clicked point.
     * @param yLoc the y location of the clicked point.
     * @return true if the bounding box contains the clicked point.

```

```

    */
    public boolean containsClickedPoint (int xLoc, int yLoc)
    {
        int HITDISTANCE = 10;
        Vector points = edge.getPoints ();

        for (Enumeration enum = points.elements (); enum.hasMoreElements
            ());)
        {
            Point p = (Point) enum.nextElement ();
            if (p != points.firstElement () && p != points.lastElement ())
                enum.nextElement (); // Waste the other point
            if ((Math.abs (p.x - xLoc) <= HITDISTANCE)
                && (Math.abs (p.y - yLoc) <= HITDISTANCE))
            {
                edge.setSelectedHandleIndex (points.indexOf (p));
                return true;
            }
        }

        return false;
    }

    /**
     * Update
     * s the shape by polling values from the associated Edge object.
     */
    public void update ()
    {
        edge.correctEndingPoints ();

        Vector points = edge.getPoints ();

        Point p1;
        Point p2;
        shape.reset ();

        for (Enumeration enum = points.elements (); enum.hasMoreElements
            ());)
        {
            p1 = (Point) enum.nextElement ();
            //debug: no such element exception 8/14/00 SYT
            /*
                if (p1.equals (points.firstElement ()))
                {
                    shape.moveTo (p1.x, p1.y);
                    p2 = (Point) enum.nextElement ();
                }
                else if (p1.equals (points.lastElement ()))
                {
                    p2 = p1;
                }
                else
                {
                    p2 = (Point) enum.nextElement ();
                }
            */
            if (p1.equals (points.firstElement ())
                & !p1.equals (points.lastElement ()))
            {
                shape.moveTo (p1.x, p1.y);
                p2 = (Point) enum.nextElement ();

            }
            else if (p1.equals (points.lastElement ()))
            {
                p2 = p1;
            }
            else
            {
                p2 = (Point) enum.nextElement ();
            }

            shape.quadTo (p1.x, p1.y, p2.x, p2.y);
        }
        p2 = (Point) points.lastElement ();
        p1 = (Point) points.elementAt (points.size () - 2);
        buildArrowHead (p2, p1);
    }

```

```

/**
 * Creates an arrow head for the stream.
 *
 * @param last the point before the ending point of the stream.
 * @param end the last point of the stream.
 */
public void buildArrowHead (Point last, Point end)
{
    double ARROWANGLE = 25.0;
    double ARROWSIDELENGTH = 15.0;
    double angle, tempAngle;
    double halfArrowAngle = ARROWANGLE / 2.0 * Math.PI / 180.0;

    if (last.x == end.x)
    {
        if (last.y > end.y)
            angle = Math.PI / 2.0;
        else
            angle = 3.0 * Math.PI / 2;
    }
    else
    {
        angle = Math.atan ((double) (last.y - end.y)
                           / (double) (last.x - end.x));
        if (last.x < end.x)
            angle = Math.PI + angle;
    }
    tempAngle = angle - halfArrowAngle;
    shape.lineTo (last.x - (int) (Math.cos (tempAngle) *
                                         ARROWSIDELENGTH),
                  last.y - (int) (Math.sin(tempAngle) *
                                         ARROWSIDELENGTH));
    tempAngle = angle + halfArrowAngle;
    shape.lineTo (last.x - (int) (Math.cos (tempAngle) *
                                         ARROWSIDELENGTH),
                  last.y - (int) (Math.sin(tempAngle) *
                                         ARROWSIDELENGTH));
    shape.lineTo (last.x, last.y);
}

/**
 * Returns the vector that contains the handles of the shape.
 *
 * @return the vector that contains the handles of the shape
 */
public Vector getHandles ()
{
    Vector v = new Vector ();
    Vector points = edge.getPoints ();
    Point p;
    int i = HANDLESIZE / 2;
    for (Enumeration enum = points.elements (); enum.hasMoreElements
         ();)
    {
        p = (Point) enum.nextElement ();
        if (p != points.firstElement () && p != points.lastElement ())
            enum.nextElement (); // Waste the other point
        v.add (new Rectangle2D.Double (p.x - i, p.y - i, HANDLESIZE
                                     , HANDLESIZE));
    }
    return v;
}

/**
 * Deletes the Edge that is associated with this object.
 */
public void delete ()
{
    edge.delete ();
    edge = null;
    shape = null;
}

} // End of the class EdgePath
package caps.GraphEditor;

```

```

public class ColorConstants
{
    public static String COLOR_NAMES [] =
    {
        "Aqua marine", "Black", "Blue", "Blue violet", "Brown"
        , "Cadet blue", "Coral", "Cornflower blue", "Cyan", "Dark
          green"
        , "Dark olive green", "Dark orchid", "Dark slate blue"
        , "Dark slate gray", "Dark turquoise", "Dim gray", "Fire brick"
        , "Forest green", "Gold", "Golden rod", "Grey", "Green",
          "Green yellow"
        , "Indian red", "Khaki", "Light blue", "Light grey", "Light
          steel blue"
        , "Lime green", "Magenta", "Maroon", "Medium aqua marine"
        , "Medium blue", "Medium orchid", "Medium sea green"
        , "Medium slate blue", "Medium spring green", "Medium
          turquoise"
        , "Medium violet red", "Midnight blue", "Navy blue", "Orange"
        , "Orange red", "Orchid", "Pale green", "Pink", "Plum", "Red"
        , "Salmon", "Sea green", "Sienna", "Sky blue", "Slate blue"
        , "Spring green", "Steel blue", "Tan", "Thistle", "Turquoise"
        , "Violet", "Violet red", "Wheat", "White", "Yellow"
        , "Yellow green"
    };

    public static int RGB_VALUES [] =
    {
        7396243, 0, 255, 10444703, 10889770, 6266783, 16744192,
        4342383, 65535
        , 3100463, 5197615, 10040013, 7021454, 3100495, 7377883,
        5526612
        , 9315107, 2330147, 13467442, 14408560, 12632256, 65280,
        9689968
        , 5123887, 10461023, 12638681, 11053224, 9408445, 3329330,
        16711935
        , 9315179, 3329433, 3289805, 9662683, 4353858, 8323327,
        8388352
        , 7396315, 14381203, 3092303, 2302862, 16744192, 16720896,
        14381275
        , 9419919, 12357519, 15379946, 16711680, 7291458, 2330216,
        9333539
        , 3316172, 32767, 65407, 2321294, 14390128, 14204888,
        11397866, 5189455
        , 13382297, 14211263, 16777215, 16776960, 10079282
    };

};

} // End of the class ColorConstants

package caps.GraphEditor;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import caps.GraphEditor.DrawPanel;
//7/24/00
import java.util.Vector;
/**
 * The DeleteDialog allows user to confirm their deletion.
 * @author Shen-Yi Tao
 * @version 1.0
 */
public class DeleteDialog extends JDialog
{
    JPanel DeleteDialog = new JPanel();
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    JPanel jPanel2 = new JPanel();
    JLabel DeleteJLabel = new JLabel();
    JButton OKJButton = new JButton();
    JButton CancelJButton = new JButton();
    DrawPanel parentDrawPanel;

    public boolean selectedAll = false;

```

```

/* public DeleteDialog(Frame frame, String title, boolean modal)
{
    super(frame, title, modal);
} */

public DeleteDialog(DrawPanel DP, boolean isSelectAll )
{
    try
    {
        jbInit();
        pack();
    }
    catch(Exception ex)
    {
        ex.printStackTrace();
    }
    if (isSelectAll)
    {
        DeleteJLabel.setText
        ("    All deleted Operators and their child nodes will be purged
        !");
        DeleteJLabel.setFont(new java.awt.Font("Dialog", 3, 12));
    }
    else
    {
        DeleteJLabel.setText("        Deleted Operators will be purged
        !");
        DeleteJLabel.setFont(new java.awt.Font("Dialog", 3, 14));
    }
    parentDrawPanel = DP;
    selectedAll = isSelectAll;
}

void jbInit() throws Exception
{
    DeleteDialog.setLayout(borderLayout1);

    DeleteJLabel.setIcon
    (new
        ImageIcon(caps.Caps.class.getResource("Images/headImage.gif")));

    OKJButton.setFont(new java.awt.Font("Dialog", 3, 14));
    OKJButton.setPreferredSize(new Dimension(81, 33));
    OKJButton.setText("OK");
    OKJButton.addActionListener(new java.awt.event.ActionListener()
    {

        public void actionPerformed(ActionEvent e)
        {
            OKJButton_actionPerformed(e);
        }
    });
    CancelJButton.setFont(new java.awt.Font("Dialog", 3, 14));
    CancelJButton.setPreferredSize(new Dimension(81, 33));
    CancelJButton.setText("Cancel");
    CancelJButton.addActionListener(new java.awt.event.ActionListener()
    {

        public void actionPerformed(ActionEvent e)
        {
            CancelJButton_actionPerformed(e);
        }
    });
    DeleteDialog.setPreferredSize(new Dimension(420, 120));
    getContentPane().add(DeleteDialog);
    DeleteDialog.add(jPanel1, BorderLayout.NORTH);
    DeleteDialog.add(DeleteJLabel, BorderLayout.WEST);
    DeleteDialog.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(OKJButton, null);
    jPanel2.add(CancelJButton, null);

    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)

```



```

        frameSize.width = screenSize.width;
        //change 2/9/00 SYT
        setLocation((screenSize.width - frameSize.width) / 4
            , (screenSize.height - frameSize.height) / 3);
        setVisible(true);
    }

    void OKJButton_actionPerformed(ActionEvent e)
    {
        if( selectedAll)
            parentDrawPanel.deleteAllSelected();
        else
            parentDrawPanel.deleteSelected();
        dispose();
    }

    void CancelJButton_actionPerformed(ActionEvent e)
    {
        dispose();
    }
}

package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;

import java.awt.geom.*;
import java.awt.event.*;
import java.awt.print.*;
import java.util.*;
import caps.Psdl.*;
import caps.Display.*;

//add SYT
import javax.swing.tree.*;

/**
 * The drawpanel is the place where the prototypes are
 * drawn on the screen and implements the undo, redo functionality.
 * and lot of user usability issues
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class DrawPanel extends JPanel implements MouseListener
, MouseMotionListener, ActionListener,KeyListener
{
    /**
     * Store space for redo
     */
    protected Vector RedoVector;
    //add 1/2/00 SYT
    /**
     * Store space for undo
     */
    protected Vector UndoVector ;

    //add 1/28/00 SYT
    /**
     * identify if component is dragged
     */
    protected boolean draggedFlag;

    //add 1/29/00 SYT
    /**
     * identify if component is added
     */
    protected boolean addFlag;
    //add 1/29/00 SYT
    /**
     * identify if a stream is drawing
     */
    protected boolean isDrawingStream;

    //add 1/29/00 SYT
    /**

```

```

    *identify if a EXTERNAL is drawing
    */
protected boolean isExternalDrawed;
//add 1/30/00 SYT
/**
    * is mouse pressed?
    */
protected boolean isPressedOn;
/**
    * The constant width of the DrawPanel
    */
public static final int WIDTH = 1024;

/**
    * The constant height of the DrawPanel
    */
public static final int HEIGHT = 768;

private final Cursor DEFAULT_CURSOR = new Cursor
    (Cursor.DEFAULT_CURSOR);

private final Cursor HAND_CURSOR = new Cursor (Cursor.HAND_CURSOR);

private final Cursor MOVE_CURSOR = new Cursor (Cursor.MOVE_CURSOR);

/**
    * The constant which specifies an operator
    */
public final static int OPERATOR = 1;

/**
    * The constant which specifies a terminator
    */
public final static int TERMINATOR = 2;

/**
    * The constant which specifies a stream
    */
public final static int STREAM = 3;

/**
    * The value of this variable is true if the toolbar is
    *in the select mode
    */
protected boolean selectMode;

/**
    * The frame which has created this DrawPanel object
    */
protected Editor parentFrame;

/**
    * This vector holds the shapes that are drawn in the DrawPanel.
    * Each shape is redrawn in the paint method by polling them
    * from this Vector.
    */
protected Vector displayComponentVector;
/**
    * vector for handle_marker on each component. SYT
    */
protected Vector handlesVector;
/**
    *this is only for single selection 1/6/00 SYT
    */
protected DisplayComponent selectedComponent;
//add 1/30/00 SYT
/**
    * current display external
    */
protected DisplayExternal currentExternal;

protected boolean MOVING_COMPONENT = false;

protected boolean MOVING_LABEL = false;

protected boolean MOVING_MET = false;

```

```

protected boolean RESIZING = false;

protected boolean IS_COLLECTING_POINTS = false;

protected boolean MOVING_ALL = false;

protected Point2D diagonalPoint;

protected VertexProperties vPropertyPanel;

protected EdgeProperties ePropertyPanel;

protected Vertex parentVertex; // The parent of the current Level

protected EdgePath currentEdge;

protected boolean selectionDefault;

/**
 * Current component is either an OPERATOR, or a TERMINATOR or a
 * STREAM
 * according to the selection from the toolbar.
 */
protected int currentComponent;

protected Popup popupMenu;

protected boolean selectAllMode;

protected Point prevPoint;

protected Rectangle bounds;

protected int currentColor;

protected int currentFont;

/**
 * Constructs a new ToolBar object
 *
 * @param frame The parent frame of this DrawPanel object.
 */
public DrawPanel (Editor frame, Vertex root)
{
    super();

    //add 2/3/00 SYT
    requestFocus();

    //add 1/2/00 SYT
    // redo vector
    RedoVector=new Vector(1);

    //add 1/2/00 SYT
    UndoVector=new Vector(1);

    //add 1/29/00 SYT
    //is now drawing a stream?
    isDrawingStream=false;
    //add 1/28/00 SYT
    //is mouse dragged?
    draggedFlag = false;
    //add 1/29/00 SYT
    // is component added?
    addFlag = false;
    //add 1/29/00 SYT
    // is External drawn?
    isExternalDrawed=false;
    //add 1/30/00 SYT
    // current focused Enternal
    currentExternal=null;
    //add 1/30/00 SYT
    // is mouse pressed on component?
    isPressedOn =false;

    popupMenu = new Popup (this);

```

```

    parentFrame = frame;

    setAlignmentX (LEFT_ALIGNMENT); // Panel does not accept these
    setAlignmentY (TOP_ALIGNMENT);
    setBorder (BorderFactory.createEtchedBorder ());

    selectMode = true;    // Initially in the selectMode.

    selectedComponent = null;

    currentEdge = null;

    vPropertyPanel = new VertexProperties (frame);
    ePropertyPanel = new EdgeProperties (frame);
    vPropertyPanel.setVisible (false);
    ePropertyPanel.setVisible (false);

    TextEditor editor = new TextEditor (frame);
    IdListEditor idEditor = new IdListEditor (frame);

    displayComponentVector = new Vector ();
    handlesVector = new Vector ();

    parentVertex = root;    // This is the root

    diagonalPoint = null;

    setCursor (DEFAULT_CURSOR);

    selectAllMode = false;
    selectionDefault = false;

    prevPoint = new Point ();

    bounds = null;

    currentColor = 61;    // White; 61 because it is index
    currentFont = 4;    // Courier Plain 12 (it is 5 - 1 )

    addMouseListener (this);    // Register mouse events
    addMouseMotionListener (this);
    addKeyListener(this);
}

//add 2/3/00 SYT
/**
 * is component selected
 */
public boolean isDFCSelected()
{
    if(selectedComponent == null)
        return false;
    else
        return true;
}

//add 1/16/00 SYT
/**
 * push tree model to undo vector
 */
public void pushUndoVector()
{
    DefaultTreeModel DTM =
        parentFrame.getTreePanel().cloneTreeModel();

    UndoVector.addElement(DTM);
    parentFrame.getEditorMenuBar().getEditMenu().getUndoMenuItem()
        .setEnabled(true);
    parentFrame.getToolBar().getUndoButton().setEnabled(true);
}

//add 1/16/00 SYT
/**
 * push tree model to redo vector
 */
public void pushRedoVector()
{
    DefaultTreeModel DTM =

```

```

    parentFrame.getTreePanel().cloneTreeModel();
    RedoVector.addElement(DTM);
    parentFrame.getEditorMenuBar().getEditMenu().getRedoMenuItem()
        .setEnabled(true);
    parentFrame.getToolBar().getRedoButton().setEnabled(true);
}

//add SYT 1/16/00
/**
 * pop from undo vector
 */
public void popUndoVector()
{
    if(! UndoVector.isEmpty())
    {
        UndoVector.removeElement(UndoVector.lastElement());
    }
    //set undo to false. SYT
    if ( UndoVector.isEmpty() )
    {
        parentFrame.getEditorMenuBar().getEditMenu().getUndoMenuItem()
            .setEnabled(false);
        parentFrame.getToolBar().getUndoButton().setEnabled(false);
    }
}

//add SYT 1/16/00
/**
 * pop from redo vector
 */
public void popRedoVector()
{
    if(! RedoVector.isEmpty())
    {
        RedoVector.removeElement(RedoVector.lastElement());
    }

    //set undo to false. SYT
    if ( RedoVector.isEmpty() )
    {
        parentFrame.getEditorMenuBar().getEditMenu().getRedoMenuItem()
            .setEnabled(false);
        parentFrame.getToolBar().getRedoButton().setEnabled(false);
    }
}

//add 1/16/00 SYT
/**
 * used before setParentVertex()
 * update tree model
 */
public void updateTreePanel(DefaultTreeModel DTM)
{
    parentFrame.getTreePanel().updateTreeModel(DTM);
}

//add 2/9/00 SYT
/**
 * find the parent vertex in the generated clone tree model
 */
public Vertex findCloneParentVertex(DefaultTreeModel DTM)
{
    for(Enumeration e = (DefaultMutableTreeNode )( DTM.getRoot() )
        .breadthFirstEnumeration(); e.hasMoreElements() ; )
    {
        DefaultMutableTreeNode DMTN =
            (DefaultMutableTreeNode )(
                e.nextElement() );
        if( (DMTN instanceof Vertex) && !(DMTN instanceof External) )
        {
            if( ( (Vertex)DMTN ).getIsParent() )
            {
                return (Vertex)DMTN;
            }
        }
    }
}

```

```

    }
    return (Vertex) (DTM.getRoot());
}

//add. 1/16/00 SYT
/**
 * paint screen for undo
 */
public void undoPaint()
{
    //for debug SYT
    if(UndoVector.isEmpty() )
    {
        System.out.println("Error! Undo Vector is empty.");
    }
    else
    {
        pushRedoVector() ;
        DefaultTreeModel model =
            (DefaultTreeModel) (UndoVector.lastElement());

        updateTreePanel( model );
        //update clone parentVertex SYT
        DefaultTreeModel currentModel = parentFrame.getTreePanel()
            .getTreeModel() ;
        parentVertex = findCloneParentVertex(currentModel);
        //test
        if(parentVertex == null)
        {
            System.out.println("parentVertex == null");
        }

        //add 2/15/00 SYT
        setTreePath();
        setParentVertex(parentVertex ,null);

        selectAllMode=false;
        //add 1/16/00 SYT
        setSelectMode (true);
        popUndoVector();
    }
    //set undo disable. SYT
    if(UndoVector.isEmpty())
    {
        parentFrame.getEditorMenuBar().getEditMenu()
            .getUndoMenuItem().setEnabled(false);

        parentFrame.getToolBar().getUndoButton().setEnabled(false);
    }
}

//add. 1/16/00 SYT
/**
 * paint screen for redo
 */
public void redoPaint()
{
    //for debug SYT
    if(RedoVector.isEmpty() )
    {
        System.out.println("Error!Redo Vector is empty.");
    }
    else
    {
        pushUndoVector() ;
        //updateTreePanel( (DefaultTreeModel) (RedoVector.lastElement()) );
        DefaultTreeModel model =
            (DefaultTreeModel) (RedoVector.lastElement());
        updateTreePanel(model);
        //update clone parentVertex SYT
        DefaultTreeModel currentModel = parentFrame.getTreePanel()
            .getTreeModel() ;
        //update clone parentVertex SYT
        parentVertex = findCloneParentVertex(currentModel);

        //add 2/15/00 SYT
        setTreePath();
    }
}

```

```

        setParentVertex(parentVertex ,null);

        selectAllMode=false;
        //add 1/16/00 SYT
        setSelectMode (true);
        popRedoVector();
    }
    //set redo disable. SYT
    if(RedoVector.isEmpty())
    {
        parentFrame.getEditorMenuBar().getEditMenu()
            .getRedoMenuItem().setEnabled(false);

        parentFrame.getToolBar().getRedoButton().setEnabled(false);
    }
}
//add 1/29/00 SYT
/**
 * get current selected edge.
 */
public EdgePath getCurrentEdge()
{
    return currentEdge;
}
// add 2/15/00 SYT
/**
 *set path to parent vertex
 */
public void setTreePath()
{
    DefaultMutableTreeNode selectedParent =
        ((DefaultMutableTreeNode)parentVertex);
    //set path of the selected component
    TreeNode [] obj= selectedParent.getPath();
    TreePath TP = new TreePath(obj);
    parentFrame.getTreePanel().setSelectionPath(TP);
    //ensure selected path visible
    parentFrame.getTreePanel().expandPath(TP);
}

// modify 2/14/00 SYT
/**
 * Sets the select mode to true or false. The panel is generally in
 * the select mode unless another button is pressed in the toolbar.
 * Update tree panel
 * @param mode true if the panel is going to be in the select mode.
 */
public void setSelectMode (boolean mode)
{
    selectMode = mode;

    if (selectMode == false && selectedComponent != null)
    {
        selectedComponent = null;
        eraseHandles ();
    }
}
// 11/19/99 SYT
/**
 * go to root node
 */
public void gotoRoot ()
{
    //add 8/22/00 SYT
    //parentFrame.getToolBar ().setTerminatorButton(true);

    TreeNode [] obj;
    if (parentVertex.isRoot () == false)
    {
        setParentVertex ((Vertex) parentVertex.getRoot (), null);
        obj = ((DefaultMutableTreeNode)parentVertex.getRoot())
            .getPath();
    }
    else
    {

```

```

        obj= ((DefaultMutableTreeNode)parentVertex).getPath();
    }

    TreePath TP = new TreePath(obj);
    parentFrame.getTreePanel().setSelectionPath(TP);
    //ensure selected path visible
    parentFrame.getTreePanel().expandPath(TP);

    parentFrame.getToolBar ().setOperatorButton (true);
}

// redesign 2/22/00 SYT
/**
 * go to parent
 * this function is closed ; leaving for reference
 */
public void gotoParent ()
{
    if( displayComponentVector.isEmpty())
    {
        DefaultMutableTreeNode upperLevelParent = parentVertex ;
        // If this is not the root of this tree
        setParentVertex ((Vertex)upperLevelParent , null);
        parentFrame.getToolBar ().setOperatorButton (true);
        //set path of the selected component
        TreeNode [] obj= upperLevelParent.getPath();
        TreePath TP = new TreePath(obj);
        parentFrame.getTreePanel().setSelectionPath(TP);
        parentFrame.getTreePanel().expandPath(TP);
    }
    if(parentVertex.isRoot())
    {
        //set path of the selected component
        TreeNode [] obj= parentVertex.getPath();
        TreePath TP = new TreePath(obj);
        parentFrame.getTreePanel().setSelectionPath(TP);
        parentFrame.getTreePanel().expandPath(TP);
    }

    if (!parentVertex.isRoot())*&&
    !((DefaultMutableTreeNode)(parentVertex.getParent()).isRoot())*/)
    {
        int colonVertexID = parentVertex.getCloneVertexID();

        DefaultMutableTreeNode upperLevelParent =
            (DefaultMutableTreeNode)
                (parentVertex.getParent() ) ;

        //handel on parentVertex 2/21/00 SYT

        // paint (getGraphics ());
        // If this is not the root of this tree SYT
        setParentVertex ((Vertex)upperLevelParent , null);
        parentFrame.getToolBar ().setOperatorButton (true);
        //set path of the selected component SYT
        TreeNode [] obj= upperLevelParent.getPath();
        TreePath TP = new TreePath(obj);
        parentFrame.getTreePanel().setSelectionPath(TP);
        parentFrame.getTreePanel().expandPath(TP);
    }
}

/** redesign 3/7/00 SYT
 */
 * create a health decompose
 * call treepanel valuechange() event
 * to make a decompse
 */
public void decompose ()
{
    //1/6/00 SYT
    if (selectedComponent==null)
        return;

```



```

        //set path of the selected component
        TreeNode [] obj=

        ((DefaultMutableTreeNode)selectedComponent.getDataFlowComponent())
        .getPath();
        TreePath TP = new TreePath(obj);
        parentFrame.getTreePanel().setSelectionPath(TP);
        //ensure selected path visible
        parentFrame.getTreePanel().expandPath(TP);
    }
    //redesign 8/22/00 SYT
    /**
     * Invoked from the treepanel
     * Invoked when the focused node on the treePanel has been changed.
     * @ parent- parent vertex
     */
    public void changeLevel (Vertex parent)
    {
        setParentVertex (parent, null);
        if (parent.isTerminator ())
            parentFrame.getToolBar ().setOperatorButton (false);

        else
            parentFrame.getToolBar ().setOperatorButton (true);
    }
    // redesign 2/21/00 SYT
    /**
     * go to one upper level screen
     * @param v :upper level parent vertex
     * @param g2d : Graphics2D Object for drawing current screen.
     */
    public void setParentVertex (Vertex v, Graphics2D g2D)
    {
        //reuse this if need a gotoParent button close at 2/28/00 SYT
        //add 2/21/00 SYT
        /* if(v.isRoot() | selectedComponent != null)
        {

        parentFrame.getToolBar().getGoToParentButton().setEnabled(false);
        }
        else
        {

        parentFrame.getToolBar().getGoToParentButton().setEnabled(true);
        }*/
        //add 3/10/00 SYT
        //reset the current parent Vertex on this screen
        // resetParentVertex(v);

        setSelectMode (false);
        parentVertex = v;

        if (g2D == null)
            g2D = (Graphics2D) getGraphics ();

        displayComponentVector.removeAllElements ();

        for (Enumeration enum = parentVertex.children ()
            ; enum.hasMoreElements ();)
        {
            DataFlowComponent dfc = (DataFlowComponent) enum.nextElement
            ();
            if (dfc instanceof Vertex && !(((Vertex) dfc).isTerminator ())
                &&
                !(dfc instanceof External))
            {
                DisplayVertex op = new DisplayVertex ((Vertex) dfc);
                op.setLabelShape (g2D);
                op.setMetShape (g2D);
                //add element
                displayComponentVector.addElement (op);
            }
            else if (dfc instanceof Vertex && (((Vertex) dfc).isTerminator
            ())
                && !(dfc instanceof External))

```

```

        {
            DisplayVertex tr = new DisplayVertex ((Vertex) dfc);
            tr.setLabelShape (g2D);
            tr.setMetShape (g2D);
            displayComponentVector.addElement (tr);
        }
        else
        {
            EdgePath ep = new EdgePath ((Edge) dfc);
            ep.setLabelShape (g2D);
            ep.setMetShape (g2D);
            ((Edge) ep.getDataFlowComponent ().correctLabelOffset ();

            ep.update ();
            displayComponentVector.addElement (ep);

            if (((Edge) dfc).getSource () instanceof External)
            {
                DisplayExternal extern =
                    new DisplayExternal ((External)((Edge)
                        dfc).getSource ());
                extern.setLabelShape ((Graphics2D) getGraphics ());
                extern.setMetShape ((Graphics2D) getGraphics ());
                displayComponentVector.addElement (extern);
            }
            else if (((Edge) dfc).getDestination () instanceof
                External)
            {
                DisplayExternal extern =
                    new DisplayExternal ((External)((Edge)
                        dfc).getDestination ());
                extern.setLabelShape ((Graphics2D) getGraphics ());
                extern.setMetShape ((Graphics2D) getGraphics ());
                displayComponentVector.addElement (extern);
            }
        }
    }

    clearAllComponentsFromScreen (g2D);
    paint (g2D);
    setSelectMode (true);
}

//modify 2/14/00 SYT
/**
 * erase the selected maker
 */
public void eraseHandles ()
{
    handlesVector.removeAllElements ();
    clearAllComponentsFromScreen (null);
    paint (getGraphics ());
}

//add SYT 12/31/99
/**
 * clearAllComponent usefull for only refresh this screen.
 */
public void clearAllComponentsFromScreen (Graphics2D g2D)
{
    if (g2D == null)
        g2D = (Graphics2D) getGraphics ();
    g2D.setColor (Color.white);
    g2D.fillRect (0, 0, WIDTH, HEIGHT);
}

/**
 * Sets the currentComponent variable to the specified argument.
 * @param component OPERATOR, TERMINATOR or STREAM
 */
public void setCurrentComponent (int component)
{
    currentComponent = component;
}

//redesign 1/24/00 SYT
/**
 * mark the selected DFC and notify the tree panel

```

```

    * @param dfc selected component
    */
    public void setSelectedDFC (DataFlowComponent dfc)
    {
        DisplayComponent dc;
        for (Enumeration enum = displayComponentVector.elements ()
            ; enum.hasMoreElements
            ());
        {
            dc = (DisplayComponent) enum.nextElement ();
            //find the selected dfc from the display vector and mark it.
            if (dc.getDataFlowComponent ().equals (dfc))
            {
                selectedComponent = dc;
                handlesVector = dc.getHandles ();
                paint (getGraphics ());
            }
        }
    }
    //SYT
    /**
    * Creates a new Operator and a new OperatorCircle object.
    * Calls the paintComponent () method to draw the component to this
    * panel.
    * @param xLoc The x location of the component.
    * @param yLoc The y location of the component.
    */
    public void processOperator (int xLoc, int yLoc)
    {
        Graphics2D g2D = (Graphics2D) getGraphics ();
        if (selectionDefault)
            setSelectionMode (true); // It will allow to place only one
            component
            // at a time SYT
            //generate new Vertex. SYT
            Vertex op = new Vertex (xLoc, yLoc, parentVertex, false);
            op.setColor (currentColor + 1);
            op.setLabelFontIndex (currentFont + 1);
            op.setMetFontIndex (currentFont + 1);
            //add to tree. 1/9/00 SYT
            parentFrame.getTreePanel().addNewDFC (op, parentVertex);
            DisplayVertex opCircle = new DisplayVertex (op);
            opCircle.setLabelShape (g2D);
            opCircle.setMetShape (g2D);
            paintComponent (opCircle);
            displayComponentVector.addElement (opCircle);
        }

        public void processTerminator (int xLoc, int yLoc)
        {
            Graphics2D g2D = (Graphics2D) getGraphics ();
            if (selectionDefault)
                setSelectionMode (true); // It will allow to place only one
                component
                //at a time
                Vertex term = new Vertex (xLoc, yLoc, parentVertex, true);
                term.setColor (currentColor + 1);
                term.setLabelFontIndex (currentFont + 1);
                term.setMetFontIndex (currentFont + 1);
                //add to tree. 1/9/00 SYT
                parentFrame.getTreePanel ().addNewDFC (term, parentVertex);
                DisplayVertex tRectangle = new DisplayVertex (term);
                tRectangle.setLabelShape (g2D);
                tRectangle.setMetShape (g2D);
                paintComponent (tRectangle);
                displayComponentVector.addElement (tRectangle);
            }
            //SYT
            /**
            * errors and too simple ,redesign is needed.
            * first find source/destination
            * check if this is the second or more click
            */
            public void processStream (int x, int y, int clicks)
            {
                isDrawingStream = true;
                DisplayComponent dc;

```

```

Vertex v = null;
//find destination vertex SYT
for (Enumeration enum = displayComponentVector.elements ();
enum.hasMoreElements ());
{
    dc = (DisplayComponent) enum.nextElement ();

    if (dc instanceof DisplayVertex && dc.getShape ().contains (x,
        y))
        v = (Vertex) dc.getDataFlowComponent ();
}
//if statement has been modified 1/29/00 SYT
if (IS_COLLECTING_POINTS) // Second or more click    SYT
{
    //find the destination SYT
    if (v != null)
    {
        //add 2/3/00 SYT
        //clearn the stream trace on the screen
        currentEdge.reset();
        // Found the destination
        v.addInEdge ((Edge) currentEdge.getDataFlowComponent ());
        ((Edge) currentEdge.getDataFlowComponent ().setDestination
        (v);
        ((Edge) currentEdge.getDataFlowComponent ().addPoint (x,
        y);
        parentFrame.getTreePanel ().addNewDFC ((Edge)
        currentEdge.getDataFlowComponent (), parentVertex);
        currentEdge.setLabelShape ((Graphics2D) getGraphics ());
        currentEdge.setMetShape ((Graphics2D) getGraphics ());
        ((Edge) currentEdge.getDataFlowComponent
        ()).correctLabelOffset ();
        currentEdge.update ();
        IS_COLLECTING_POINTS = false;
        //add EdgePath to the display vector. SYT
        displayComponentVector.addElement (currentEdge);
        if (selectionDefault)
        {
            parentFrame.getToolBar ().enableSelectButton ();
            setSelectMode (true);
        }
        //add 1/29/00 SYT
        //finish drawing
        isDrawingStream=false ;
        //add 1/30/00 SYT
        isExternalDrawed=false;
        //change 2/4/00 SYT
        // paintComponent (currentEdge);
        clearAllComponentsFromScreen (null);
        paint(getGraphics());
    }
    else
    { // collect the next point
        ((Edge) currentEdge.getDataFlowComponent ().addPoint (x,
        y);
        //add 2/3/00 SYT
        //draw line SYT
        currentEdge.quadTo(x,y);
        drawSegment(currentEdge.getShape());
    }
}
//1/29/00 SYT
//stream source:external
else if (v == null)
{ // This is an external -> vertex stream
    //add 1/29/00 SYT
    pushUndoVector();
    //add 1/29/00 SYT
    isExternalDrawed=true;
    IS_COLLECTING_POINTS = true;
    //create new external SYT
    External ex = new External (x, y, parentVertex);
    //create new edge SYT
    Edge ed = new Edge (x, y, parentVertex);

    ed.setLabelFontIndex (currentFont + 1);
    ed.setMetFontIndex (currentFont + 1);
}

```

```

        ex.setLabelFontIndex (currentFont + 1);
        ex.setMetFontIndex (currentFont + 1);
        //add out edge SYT
        ex.addOutEdge (ed);
        //set destination SYT
        ed.setSource (ex);
        //create edge path SYT
        currentEdge = new EdgePath (ed);

        //add 2/3/00 SYT
        //draw segment
        currentEdge.quadTo(x,y);
        drawSegment(currentEdge.getShape());

        //create display extern SYT
        DisplayExternal extern = new DisplayExternal (ex);
        //add 1/30/00 SYT
        currentExternal=extern;
        extern.setLabelShape ((Graphics2D) getGraphics ());
        extern.setMetShape ((Graphics2D) getGraphics ()); //
        *** Maybe we don't need this *****
        //display component SYT
        displayComponentVector.addElement (currentExternal);
        paintComponent (currentExternal);
    }
    //first clicled SYT
    // source : vertex
    else
    {
        //add 1/29/00 SYT
        pushUndoVector();
        // First click // vertex-vertex or vertex-external
        IS_COLLECTING_POINTS = true;
        Edge ed = new Edge (x, y, parentVertex);

        ed.setLabelFontIndex (currentFont + 1);
        ed.setMetFontIndex (currentFont + 1);
        v.addOutEdge (ed);
        ed.setSource (v);
        currentEdge = new EdgePath (ed);
        //add 2/3/00 SYT
        //draw segment
        currentEdge.quadTo(x,y);
        drawSegment(currentEdge.getShape());
    }
}
// add 2/3/00 SYT
/**
 * draw segment between two points
 */
public void drawSegment(Shape gp)
{
    Shape shape = gp;

    Graphics2D g2D = (Graphics2D) getGraphics ();
    // g2D.setColor (new Color (ColorConstants.RGB_VALUES
    [currentColor]));
    g2D.fill (shape);
    g2D.setColor (Color.blue);
    g2D.draw (shape);
}

/**
 * Paints the component into this panel by calling the
 * graphics2D.draw(Shape) method.
 * @param component The component to be drawn into the panel
 */
public void paintComponent (DisplayComponent component)
{
    Graphics2D g2D = (Graphics2D) getGraphics ();

    if (component instanceof DisplayVertex)
    {
        g2D.setColor (new Color (ColorConstants.RGB_VALUES
        [currentColor]));
        //g2D.fill (((DisplayVertex) component).getPaintedShape ());
    }
}

```

```

        g2D.fill (component.getShape ());
        g2D.setColor (Color.black);
    }
    g2D.draw (component.getShape ());
    component.drawLabelShape (g2D);
    if (component instanceof DisplayVertex)
        component.drawMetShape (g2D);
}

/**
 * This method is called to repaint
 * all the components when necessary.
 * @param g The graphics context of the panel
 */
public void paint (Graphics g)
{
    //modified 2/15/00 SYT
    Graphics2D g2D;

    if(g == null)
    {
        g2D = (Graphics2D) getGraphics();
    }
    else
    {
        g2D = (Graphics2D) g;
    }
    g2D.setColor (Color.black);

    for (Enumeration e = displayComponentVector.elements ()
        ; e.hasMoreElements ();)
    {
        DisplayComponent dcp = (DisplayComponent) e.nextElement ();
        DataFlowComponent dfc = dcp.getDataFlowComponent ();
        if (MOVING_COMPONENT
            || (MOVING_LABEL && selectedComponent.getDataFlowComponent ()
                instanceof External))
        {
            dcp.update (); // and also label changes vs
            if (dfc instanceof Edge && ((Edge) dfc).isStateStream ())
            {
                g2D.setStroke (new BasicStroke (1.5f));
                g2D.draw (dcp.getShape ());
                g2D.setStroke (new BasicStroke (1f));
            }
            else
            {
                if (dcp instanceof DisplayVertex)
                {
                    g2D.setColor (new Color (ColorConstants.RGB_VALUES
                        [((Vertex) dfc).getColor () - 1]));
                    g2D.fill (((DisplayVertex) dcp).getPaintedShape ());
                    g2D.setColor (Color.black);
                }
                g2D.draw (dcp.getShape ());
                if (!dfc.isLeaf ())
                    g2D.draw (((DisplayVertex) dcp).getInnerShape ());
            }
            dcp.drawLabelShape (g2D);
            dcp.drawMetShape (g2D);
        }
        if ((selectMode && selectedComponent != null) || selectAllMode)
        {
            //mark the component. SYT 12/31/99
            for (Enumeration e = handlesVector.elements ();
                e.hasMoreElements ();)
            {
                g2D.setColor (Color.gray);
                g2D.fill ((Shape) e.nextElement ());
            }
        }
    }
}

/**
 * Sets the size of the panel to WIDTH and HEIGHT
 * @return Returns a new Dimension object initialized to the
 *         WIDTH and HEIGHT parameters.
 */

```

```

public Dimension getPreferredSize ()
{
    return new Dimension (WIDTH, HEIGHT);
}

// redesign SYT
/**
 * Handles the event that occurs when a mouse button is clicked on
 * this panel
 * @param e The MouseEvent that occurs.
 */
public void mousePressed (MouseEvent e)
{
    int xPosition = e.getX ();
    int yPosition = e.getY ();

    int flags = e.getModifiers ();
    prevPoint.setLocation (xPosition, yPosition);

    //add SYT 12/31/99
    if ( e.isControlDown() )
    {
        //hot key not implemented yet. SYT
        //System.out.println("ControlDown");
    }
    //left mouse button or right mouse button was pressed. SYT
    //12/28/99
    else if (flags == MouseEvent.BUTTON1_MASK
        || flags == MouseEvent.BUTTON3_MASK)
    {
        if (!selectAllMode && isHoldingHandle
            (xPosition, yPosition))
        {
            //set RESIZING.(draggable) 1/12/00 SYT
            if (selectedComponent instanceof DisplayVertex &&
                selectedComponent.getShape ().getBounds2D ()
                    .contains (diagonalPoint)) // Make sure it is not the
                label
                //resize component SYT
                RESIZING = true;
        }
        else if (selectMode)
        {
            DisplayComponent dc;
            boolean flag = false;
            for (Enumeration enum = displayComponentVector.elements ()
                ; enum.hasMoreElements ();)
            {
                dc = (DisplayComponent) enum.nextElement ();
                if (selectAllMode
                    && (dc.containsClickedPoint (xPosition, yPosition)
                        || dc.getLabelShapeBounds ().contains (xPosition,
                            yPosition)
                        || (dc.getMetShapeBounds ().contains (xPosition,
                            yPosition))))
                {
                    MOVING_ALL = true;
                    flag = true;
                }
            }
            //paint selected DataFlowComponent. 1/12/00 SYT
            else if (dc.getLabelShapeBounds ()
                .contains (xPosition,
                    yPosition))
            { // If clicked a label
                eraseHandles ();

                //add 2/4/00 SYT
                if( !(dc.getDataFlowComponent() instanceof External))
                {
                    //add 2/28/00 SYT
                    //Both of the parent and child focused
                    //on the parent itself
                    TreeNode [] obj=
                        ((DefaultMutableTreeNode)
                            ((DefaultMutableTreeNode)

```

```

(dc.getDataFlowComponent()).getParent().getPath();
    //add 2/12/00 SYT
    TreePath TP = new TreePath(obj);
    //add 2/12/00 SYT
    //ensure selected path visible
    parentFrame.getTreePanel().expandPath(TP);

}
//if(parentFrame.getTreePanel().isPathSelected(TP))
//parentFrame.getTreePanel().getTreeModel().reload();

//reset marker SYT
handlesVector =
dc.getStringHandles(dc.getLabelShapeBounds());
selectedComponent = dc;
clearAllComponentsFromScreen((Graphics2D)
getGraphics());
paint (getGraphics ());
flag = true;

}
else if (dc.getMetShapeBounds ().contains (xPosition,
yPosition))
{ // If clicked an met
    eraseHandles ();

    //add 2/4/00 SYT
    if( !(dc.getDataFlowComponent() instanceof External)
    )
    {
        //add 2/15/00 SYT
        /*if(dc.getDataFlowComponent() instanceof Vertex)
        {
            parentFrame.getToolBar().getDecomposeButton()
            .setEnabled(true);
        }*/

        //changed on 2/28/00 SYT
        /*if( ((DefaultMutableTreeNode)
(dc.getDataFlowComponent()))
        .isLeaf() )
        {
            //set path of the selected component
            TreeNode [] obj= ((DefaultMutableTreeNode)
            {dc.getDataFlowComponent() }).getPath();
            //add 2/12/00 SYT
            //following lines will make labe marker disapear
            TreePath TP = new TreePath(obj);
            parentFrame.getTreePanel().setSelectionPath(TP);
            //add 2/12/00 SYT
            //ensure selected path visible
            parentFrame.getTreePanel().expandPath(TP);
        }
        // if parent cancel the focuse
        else
        {
            //set path of the selected component
            TreeNode [] obj=
            ((DefaultMutableTreeNode)
            ((DefaultMutableTreeNode)
(dc.getDataFlowComponent()).getParent().getPath();
            //add 2/12/00 SYT
            TreePath TP = new TreePath(obj);
            //add 2/12/00 SYT
            //ensure selected path visible
            parentFrame.getTreePanel().expandPath(TP);

            parentFrame.getTreePanel().clearSelection();
        } */
        //add 2/28/00 SYT
        //Both of the parent and child focused
        //on the parent itself
        TreeNode [] obj=
        ((DefaultMutableTreeNode)
        ((DefaultMutableTreeNode)

```



```

(dc.getDataFlowComponent()).getParent().getPath();
    //add 2/12/00 SYT
    TreePath TP = new TreePath(obj);
    //add 2/12/00 SYT
    //ensure selected path visible
    parentFrame.getTreePanel().expandPath(TP);
}

//if(parentFrame.getTreePanel().isPathSelected(TP))
//parentFrame.getTreePanel().getTreeModel().reload();
//reset marker SYT
handlesVector = dc.getStringHandles
(dc.getMetShapeBounds ());

selectedComponent = dc;
paint (getGraphics ());
flag = true;
}
else if (dc.containsClickedPoint (xPosition, yPosition))
{ // If clicked on a component
    eraseHandles ();
    //add 2/4/00 SYT
    if( !(dc.getDataFlowComponent() instanceof External ) )
    {
        //add 2/15/00 SYT
        /* if(dc.getDataFlowComponent() instanceof Vertex)
        {
            parentFrame.getToolBar().getDecomposeButton()
            .setEnabled(true);
        }*/

        /* change on 2/28/00 SYT
        if( ((DefaultMutableTreeNode)
        (dc.getDataFlowComponent()) )
        .isLeaf() )
        {
            //set path of the selected component
            TreeNode [] obj= ((DefaultMutableTreeNode)
            (dc.getDataFlowComponent()) ).getPath();
            //add 2/12/00 SYT
            //following lines will make labe marker disappear
            TreePath TP = new TreePath(obj);
            parentFrame.getTreePanel().setSelectionPath(TP);
            //add 2/12/00 SYT
            //ensure selected path visible
            parentFrame.getTreePanel().expandPath(TP);
        }
        // if parent cancel the focuse
        else
        {
            //set path of the selected component
            TreeNode [] obj=
            ((DefaultMutableTreeNode)
            ((DefaultMutableTreeNode)

(dc.getDataFlowComponent()).getParent().getPath();
            //add 2/12/00 SYT
            TreePath TP = new TreePath(obj);
            parentFrame.getTreePanel().setSelectionPath(TP);
            //add 2/12/00 SYT
            //ensure selected path visible
            parentFrame.getTreePanel().expandPath(TP);

            parentFrame.getTreePanel().clearSelection();
        }*/
        //add 2/28/00 SYT
        //Both of the parent and child focused
        //on the parent itself
        TreeNode [] obj=
        ((DefaultMutableTreeNode)
        ((DefaultMutableTreeNode)

dc.getDataFlowComponent()).getParent().getPath();
        //add 2/12/00 SYT
        TreePath TP = new TreePath(obj);
        //add 2/12/00 SYT
        //ensure selected path visible

```

```

        parentFrame.getTreePanel().expandPath(TP);

    }
    //if(parentFrame.getTreePanel().isPathSelected(TP))
    //parentFrame.getTreePanel().getTreeModel().reload();

    //reset marker SYT
    eraseHandles();
    handlesVector = dc.getHandles ();
    selectedComponent = dc;
    paint (getGraphics ());
    flag = true; // Clicked on a component
}
if (flag)
    setCursor (MOVE_CURSOR);
}
//add 2/14/00 SYT
//mouse press on empty space
if(!flag)
{
    setTreePath();
    //add 2/15/00 SYT

    // parentFrame.getToolBar().getGoToParentButton()
    // .setEnabled(false);
}

if (selectedComponent != null && !flag)
{
    selectedComponent = null;
    eraseHandles ();
    setSelectAllMode (false);
}

//add 1/28/00 SYT
//popUndoVector may be excuted on mouseReleased
if(flag|RESIZING)
{
    pushUndoVector();
    //add SYT 1/30/00 SYT
    isPressedOn =true;
}
setMenuBarItems ();
}

//add component. SYT 1/12/00
else if (flags != MouseEvent.BUTTON3_MASK)
{
    parentVertex.setAllowsChildren (true);
    //switch OPERATOR: TERMINATOR: STREAM:

    //add 1/29/00 SYT
    addFlag = true;

    switch (currentComponent)
    {
        case OPERATOR:
            //add 1/29/00 SYT
            pushUndoVector();

            processOperator (xPosition, yPosition);
            if (selectionDefault)
                parentFrame.getToolBar ().enableSelectButton ();
            parentFrame.setSaveRequired (true);
            break;
        //error and has been changed to correct. SYT 1/12/00
        //case OPERATOR:
        case TERMINATOR:
            //add 1/29/00 SYT
            pushUndoVector();
            processTerminator (xPosition, yPosition);
            if (selectionDefault)
                parentFrame.getToolBar ().enableSelectButton ();
            parentFrame.setSaveRequired (true);
            break;
        case STREAM:
            //here,undo need be implemented in processStream()
    }
}

```

```

        //pushUndoVector();
        processStream (xPosition, yPosition, e.getClickCount
        ()); // Pending same as chriss' implementation
        parentFrame.setSaveRequired (true);
        break;
    default:
        break;
    }
}
//add 1/30/00 SYT
else
    isPressedOn =false;
}
//reuse this if need a gotoParent button close at 2/28/00 SYT
/* if(parentVertex.isRoot() | selectedComponent!=null )
{

parentFrame.getToolBar().getGoToParentButton().setEnabled(false);
}
else
{

parentFrame.getToolBar().getGoToParentButton().setEnabled(true);
}*/
}
// modify 4/19/00 SYT
/**
 * set MenuItems
 * prevent type name Vertex from decomposing
 */
public void setMenuBarItems ()
{
    if (selectedComponent == null)
    {
        // delete SYT
        parentFrame.getJMenuBar ().getMenu (1).getItem (4).setEnabled
        (false);
        // decompose SYT
        //parentFrame.getJMenuBar ().getMenu (3).getItem
        (2).setEnabled (false);
        parentFrame.getJMenuBar ().getMenu (3).getItem (1).setEnabled
        (false);
        // color SYT
        parentFrame.getJMenuBar ().getMenu (2).getItem (0).setEnabled
        (true);
    }
    else
    {
        // delete
        parentFrame.getJMenuBar ().getMenu (1).getItem (4).setEnabled
        (true);
        if (selectedComponent instanceof EdgePath)
        {
            // color
            parentFrame.getJMenuBar ().getMenu (2).getItem (0)
            .setEnabled (false);

            // decompose
            //parentFrame.getJMenuBar ().getMenu (3).getItem (2)
            // .setEnabled (false);
            parentFrame.getJMenuBar ().getMenu (3).getItem (1)
            .setEnabled (false);
        }
        else
        {
            // color
            parentFrame.getJMenuBar ().getMenu (2).getItem
            (0).setEnabled(true);
            if (!(selectedComponent instanceof EdgePath))
            {
                if (((Vertex)(selectedComponent.getDataFlowComponent()))
                )

                .getLabel().indexOf(".") == -1)
                // decompose
                //parentFrame.getJMenuBar ().getMenu (3).getItem (2)
                // .setEnabled (true);
                parentFrame.getJMenuBar ().getMenu (3).getItem (1)
                .setEnabled (true);
            }
            else

```

```

        // decompose
        //parentFrame.getJMenuBar ().getMenu (3).getItem (2)
        //                                     .setEnabled (false);
        parentFrame.getJMenuBar ().getMenu (3).getItem (1)
                                     .setEnabled (true);
    }
}

/**
 * Handles the event that occurs when the mouse enters into the
 * panel.
 *
 * @param e The MouseEvent that occurs.
 */
public void mouseEntered (MouseEvent e)
{
}

/**
 * add 1/30/00 SYT
 * Handles the event that occurs when the mouse exits the panel.
 * dress stream_drawing_unfinish problem
 * @param e The MouseEvent that occurs.
 */
public void mouseExited (MouseEvent e)
{
    clearnDrawStream();
}

/**
 * Add 2/3/00 SYT
 * deal with unfinished job after stop drawing stream
 * clean screen
 */
public void clearnDrawStream()
{
    if(!(currentEdge==null) )
    {
        if(isDrawingStream)
        {
            //delete external and it's children
            // currentEdge.delete();
            int index=parentVertex.getIndex(currentEdge.getEdge());
            ((DefaultMutableTreeNode)parentVertex).remove(index);

            displayComponentVector
            .removeElement(currentExternal);
            parentFrame.getTreePanel ().removeDfc();

            //setParentVertex (parentVertex, null);
            clearAllComponentsFromScreen (null);
            paint(getGraphics());
        }
        //vertex-> SYT
        else
        {
            //remove edge SYT

            int index=parentVertex.getIndex(currentEdge.getEdge());
            //add 2/9/00 SYT
            currentEdge.getEdge().getSource()

            .removeOutEdge(currentEdge.getEdge());
            ((DefaultMutableTreeNode)parentVertex).remove(index);

            parentFrame.getTreePanel ().removeDfc();

            clearAllComponentsFromScreen (null);
            paint(getGraphics());
        }

        if (selectionDefault)

```

```

        {
            parentFrame.getToolBar ().enableSelectButton ();
            setSelectMode (true);
        }
        //still focus on draw stream. SYT
        setSelectMode (false);
        setCurrentComponent (DrawPanel.STREAM);

        popUndoVector();
        isExternalDrawed=false;
        isDrawingStream=false;
        currentEdge=null;
        currentExternal=null;
        IS_COLLECTING_POINTS=false;
    }
}
}
//modify 4/19/00 SYT
/**
 * Handles the event that occurs when a mouse button is pressed on
 * this panel
 * prevent type-name vertex from decomposing
 * @param e The MouseEvent that occurs.
 */
public void mouseClicked(MouseEvent e)
{
    int xPosition = e.getX ();
    int yPosition = e.getY ();
    int flags = e.getModifiers ();
    int clickCount = e.getClickCount ();
    //non-External DataFlowComponent selected and not in "select all
    //mode". SYT 1/12/00
    if (selectedComponent != null &&
        !(selectedComponent.getDataFlowComponent () instanceof External)
        && !selectAllMode)
    {
        //right mouse button clicked.
        //pop up a menu.
        //SYT 12/28/99
        if (flags == MouseEvent.BUTTON3_MASK)
            if (selectedComponent instanceof EdgePath)
                popupMenu.showPopupMenu (true, xPosition, yPosition);
        // disables decompose
        else
        {
            if (((Vertex)(selectedComponent.getDataFlowComponent()))
                .getLabel().indexOf(".") == -1)
            {
                popupMenu.showPopupMenu (false, xPosition,
                    yPosition); // enables decompose
            }
            else
            {
                // disables decompose
                popupMenu.showPopupMenu (true, xPosition, yPosition);
            }
        }
        else if (clickCount > 1)
            showProperties (selectedComponent);
    }
    //vertex->external SYT
    else if (flags == MouseEvent.BUTTON3_MASK &&
        IS_COLLECTING_POINTS)
    {
        //add 4/20/00 SYT
        if( !(((Edge)currentEdge.getDataFlowComponent())
            .getSource() instanceof External) )
        {
            //add 2/3/00 SYT
            //clearn the stream trace on the screen
            currentEdge.reset();
            //drawing a stream destination is External SYT
            External ex = new External (xPosition, yPosition,
                parentVertex);
            //external add in_edge SYT
            ex.addInEdge ((Edge) currentEdge.getDataFlowComponent ());
        }
    }
}

```

```

        //set destination      SYT
        ((Edge) currentEdge.getDataFlowComponent ()).setDestination
        (ex);
        ((Edge) currentEdge.getDataFlowComponent ())
            .addPoint (xPosition, yPosition);
        parentFrame.getTreePanel ().addNewDFC ((Edge) currentEdge
            .getDataFlowComponent (), parentVertex);
        currentEdge.setLabelShape ((Graphics2D) getGraphics ());
        currentEdge.setMetShape ((Graphics2D) getGraphics ());
        ((Edge) currentEdge.getDataFlowComponent
            ()).correctLabelOffset ();
        currentEdge.update ();
        IS_COLLECTING_POINTS = false;
        displayComponentVector.addElement (currentEdge);
        if (selectionDefault)
        {
            parentFrame.getToolBar ().enableSelectButton ();
            setSelectMode (true);
        }
        DisplayExternal extern = new DisplayExternal (ex);
        extern.setLabelShape ((Graphics2D) getGraphics ());
        extern.setMetShape ((Graphics2D) getGraphics ());
        displayComponentVector.addElement (extern);
        //change 2/3/00 SYT
        //paintComponent (currentEdge);
        //paintComponent (extern);
        clearAllComponentsFromScreen (null);
        paint(getGraphics());

        //add 2/2/00 SYT
        isExternalDrawed=false;
        isDrawingStream=false;
        currentEdge=null;
        currentExternal=null;
    }
}

public void showProperties (DisplayComponent d)
{
    if (d instanceof DisplayVertex)
    {
        vPropertyPanel.setVertex ((Vertex) d.getDataFlowComponent ());
        vPropertyPanel.setDisplayVertex ((DisplayVertex) d);
    }
    else
    {
        ePropertyPanel.setEdge ((Edge) d.getDataFlowComponent ());
        ePropertyPanel.setEdgePath ((EdgePath) d);
    }
}

/**
 * Handles the event that occurs when a mouse button is released
 * on this panel
 *
 * @param e The MouseEvent that occurs.
 */
public void mouseReleased (MouseEvent e)
{
    //add 1/26/00 SYT
    if( (currentComponent != STREAM) )
    {
        if(isPressedOn)
        {
            //if(!(draggedFlag |addFlag) & !selectMode)
            if(draggedFlag |addFlag)
            {
            }
            else
            {
                popUndoVector();
            }
        }
    }
    //add 1/29/00 SYT
    isPressedOn=false;
}

```

```

draggedFlag=false;
addFlag=false;

//add 1/28/00 SYT
MOVING_COMPONENT = false;
MOVING_LABEL = false;
MOVING_MET = false;
RESIZING = false;
MOVING_ALL = false;
diagonalPoint = null;
//add 2/28/00
setSelectAllMode(false);

//setCursor (DEFAULT_CURSOR);
}

/**
 * Handles the event that occurs when the mouse is dragged on this
 * panel
 *
 * @param e The MouseEvent that occurs.
 */
public void mouseDragged (MouseEvent e)
{
    if (selectedComponent != null || MOVING_ALL)
    {
        int xPosition = e.getX ();
        int yPosition = e.getY ();
        int flags = e.getModifiers ();

        if (!MOVING_ALL)
        {
            bounds = (Rectangle) selectedComponent.getShape
                ().getBounds ();
        }
        if (((bounds.getMinX () <= 0) && (xPosition <= prevPoint.x))
            ||
            ((bounds.getMinY () <= 0) && (yPosition <= prevPoint.y))
            ||
            ((bounds.getMaxX () >= WIDTH) && (xPosition >=
                prevPoint.x)) ||
            ((bounds.getMaxY () >= HEIGHT) && (yPosition >=
                prevPoint.y)))
        {
            setCursor (DEFAULT_CURSOR);
        }
        else if (flags == MouseEvent.BUTTON1_MASK)
        {
            //NULLPOINTEXCEPTION SYT
            DataFlowComponent dfc =
                selectedComponent.getDataFlowComponent ();
            //moveing all components SYT
            if (selectAllMode && (MOVING_ALL))
            {
                handlesVector.removeAllElements ();
                bounds = (Rectangle) ((DisplayComponent)
                    displayComponentVector
                        .elementAt (0)).getShape ().getBounds ();
                for (Enumeration enum = displayComponentVector.elements
                    ())
                {
                    enum.hasMoreElements ();
                }
                selectedComponent = (DisplayComponent)
                    enum.nextElement ();
                dfc = selectedComponent.getDataFlowComponent ();
                dfc.moveTo (xPosition - prevPoint.x, yPosition -
                    prevPoint.y);
                handlesVector.addAll (selectedComponent.getHandles
                    ());
                handlesVector.addAll (selectedComponent
                    .getStringHandles
                    (selectedComponent.getLabelShapeBounds ()));
                handlesVector.addAll (selectedComponent
                    .getStringHandles
                    (selectedComponent.getMetShapeBounds ()));
            }
        }
    }
}

```

```

        selectedComponent.update ();
        bounds = (Rectangle) bounds
        .createUnion (selectedComponent.getShape ().getBounds
        ());
    }
}
else if (RESIZING)
{
    // add 1/28/00 SYT
    draggedFlag=true;
    Rectangle2D.Double r2D = (Rectangle2D.Double)
    selectedComponent
        .getShape ().getBounds2D ();
    r2D.setFrameFromDiagonal
        (new Point (xPosition, yPosition),
        diagonalPoint);
    ((Vertex) dfc).setX ((int) r2D.getCenterX ());
    ((Vertex) dfc).setY ((int) r2D.getCenterY ());
    ((Vertex) dfc).setWidth ((int) r2D.getWidth ());
    selectedComponent.update ();
    handlesVector = selectedComponent.getHandles ();
    //add 1/28/00 SYT
    parentFrame.setSaveRequired (true);
}
else if (MOVING_LABEL)
{
    // add 1/28/00 SYT
    draggedFlag=true;
    if (dfc instanceof External)
    {
        ((External) dfc).setLocation
            (xPosition - prevPoint.x, yPosition -
            prevPoint.y);
        selectedComponent.update ();
    }
    else
        dfc.setLabelOffset
            (xPosition - prevPoint.x, yPosition -
            prevPoint.y);
        handlesVector = selectedComponent
            .getStringHandles
            (selectedComponent.getLabelShapeBounds ());
}
else if (MOVING_MET)
{
    // add 1/28/00 SYT
    draggedFlag=true;
    dfc.setMetOffset
        (xPosition - prevPoint.x, yPosition - prevPoint.y);
    handlesVector = selectedComponent
        .getStringHandles (selectedComponent.getMetShapeBounds
        ());
}
else if (MOVING_COMPONENT)
{
    // add 1/28/00 SYT
    draggedFlag=true;
    if (dfc instanceof Vertex)
        ((Vertex) dfc).setLocation
            (xPosition - prevPoint.x, yPosition -
            prevPoint.y);
        else
            ((Edge) dfc).reShape (xPosition, yPosition);
        selectedComponent.update ();
        handlesVector = selectedComponent.getHandles ();
}
else
{
    if (selectedComponent.getLabelShapeBounds ()
        .contains (xPosition, yPosition))
    {
        MOVING_LABEL = true;
        if (dfc instanceof External)
        {
            ((External) dfc).setLocation
                (xPosition - prevPoint.x, yPosition -

```



```

        prevPoint.y);
        selectedComponent.update ();
    }
    else
        dfc.setLabelOffset
            (xPosition - prevPoint.x, yPosition -
prevPoint.y);
        handlesVector = selectedComponent
            .getStringHandles
(selectedComponent.getLabelShapeBounds ());
        parentFrame.setSaveRequired (true);
    }
    if (selectedComponent.getMetShapeBounds ()
        .contains (xPosition,
yPosition))
    {
        MOVING_MET = true;
        dfc.setMetOffset
            (xPosition - prevPoint.x, yPosition - prevPoint.y);
        handlesVector = selectedComponent
            .getStringHandles
(selectedComponent.getLabelShapeBounds ());
        parentFrame.setSaveRequired (true);
    }
    else if (selectedComponent
        .containsClickedPoint (xPosition, yPosition))
    {
        MOVING_COMPONENT = true;
        if (dfc instanceof Vertex)
            ((Vertex) dfc).setLocation
                (xPosition - prevPoint.x, yPosition -
prevPoint.y);
        else
            ((Edge) dfc).reShape (xPosition, yPosition);
        //selectedComponent.update ();
        handlesVector = selectedComponent.getHandles ();
        parentFrame.setSaveRequired (true);
    }
    //setCursor (MOVE_CURSOR);
}
clearAllComponentsFromScreen (null);
paint (getGraphics ());
setCursor (MOVE_CURSOR);
prevPoint.setLocation (xPosition, yPosition);
}
}
}
//add 12/28/99 SYT
/**
 *true if (x,y) in any rectangle of elements.
 *set the diagonalPoint.
 */
public boolean isHoldingHandle (int x, int y)
{
    boolean flag = false;
    Rectangle2D r2D;
    for (Enumeration e = handlesVector.elements (); e.hasMoreElements
        ());
    {
        r2D = (Rectangle2D) e.nextElement ();
        if (r2D.contains (x, y))
        {
            diagonalPoint = getDiagonalPoint (r2D);
            flag = true;
        }
    }
    return flag;
}

public Point2D getDiagonalPoint (Rectangle2D rect)
{
    Point2D p = new Point2D.Double ();
    int w = (int) rect.getWidth () / 2;
    Rectangle2D r2D = (Rectangle2D) selectedComponent
        .getShape ().getBounds
        ();
    if (rect.getMaxX () >= r2D.getMaxX ())

```

```

        && rect.getMaxY () >= r2D.getMaxY ()
        p.setLocation (r2D.getMinX () + w, r2D.getMinY () + w);
    else if (rect.getMaxX () >= r2D.getMaxX ()
        && rect.getMinY () <= r2D.getMinY ())
        p.setLocation (r2D.getMinX () + w, r2D.getMaxY () - w);
    else if (rect.getMinX () <= r2D.getMinX ()
        && rect.getMinY () <= r2D.getMinY ())
        p.setLocation (r2D.getMaxX () - w, r2D.getMaxY () - w);
    else if (rect.getMinX () <= r2D.getMinX ()
        && rect.getMaxY () >= r2D.getMaxY ())
        p.setLocation (r2D.getMaxX () - w, r2D.getMinY () + w);
    return p;
}

/**
 * Handles the event that occurs when the mouse is moved on this
 * panel
 *
 * @param e The MouseEvent that occurs.
 */
public void mouseMoved (MouseEvent e)
{
    int xPosition = e.getX ();
    int yPosition = e.getY ();
    if (selectMode)
    {
        setCursor (DEFAULT_CURSOR);
        DisplayComponent dc;
        for (Enumeration enum = displayComponentVector.elements ()
            ; enum.hasMoreElements
                ();)
        {
            dc = (DisplayComponent) enum.nextElement ();
            if (dc.containsClickedPoint (xPosition, yPosition) ||
                dc.getLabelShapeBounds ().contains (xPosition,
                    yPosition) ||
                (dc.getMetShapeBounds ().contains (xPosition, \
                    yPosition)))
            {
                if (dc.equals (selectedComponent)
                    && (MOVING_COMPONENT || MOVING_LABEL ||
                        MOVING_MET))
                    setCursor (MOVE_CURSOR);
                else
                    setCursor (HAND_CURSOR);
            }
        }
    }
    //if (IS_COLLECTING_POINTS) {
    //    rubberBandLine (prevPoint.x, prevPoint.y);
    //    rubberBandLine (xPosition, yPosition);
    //}
}

public void actionPerformed (ActionEvent e)
{
    if (e.getSource () == popupMenu.getDecomposeMenuItem ())
    {
        decompose();
    }
    if (e.getSource () == popupMenu.getFontMenuItem ())
    {
        String selected = (String) JOptionPane.showInputDialog
            (parentFrame, "Select Font : ", "Font
                Selection"
                , JOptionPane.INFORMATION_MESSAGE, null
                , FontConstants.FONT_NAMES
                ,FontConstants.FONT_NAMES [0]);
        if (selected != null)
        {
            int fontIndex = 0;
            for (int ix = 0; ix < FontConstants.FONT_NAMES.length;
                ix++)
            {
                if (FontConstants.FONT_NAMES [ix].equals (selected))

```

```

        fontIndex = ix;
    }
    selectedComponent.getDataFlowComponent ()
        .setLabelFontIndex (fontIndex + 1);
    selectedComponent.getDataFlowComponent ()
        .setMetFontIndex (fontIndex + 1);
    selectedComponent.setLabelShape ((Graphics2D) getGraphics
    ());
    selectedComponent.setMetShape ((Graphics2D) getGraphics
    ());
    clearAllComponentsFromScreen ((Graphics2D) getGraphics ());
    paint (getGraphics ());
}
}
else if (e.getSource () == popupMenu.getColorMenuItem ())
{
    String selected = (String) JOptionPane
        .showInputDialog (parentFrame, "Select
        color : "
        , "Open",
        JOptionPane.INFORMATION_MESSAGE, null
        , ColorConstants.COLOR_NAMES
        , ColorConstants.COLOR_NAMES [0]);
    if (selected != null)
    {
        int colorIndex = 0;
        for (int ix = 0; ix < ColorConstants.COLOR_NAMES.length;
            ix++)
        {
            if (ColorConstants.COLOR_NAMES [ix].equals (selected))
                colorIndex = ix;
        }
        ((Vertex) selectedComponent.getDataFlowComponent ())
            .setColor (colorIndex + 1);
        clearAllComponentsFromScreen ((Graphics2D) getGraphics ());
        paint (getGraphics ());
    }
}
else if (e.getSource () == popupMenu.getDeleteMenuItem ())
{
    deleteSelectedComponent ();
}
else if (e.getSource () == popupMenu.getPropMenuItem ())
{
    showProperties (selectedComponent);
}
}
//12/11/99 SYT
/**
 * Editor class will use this for solving key board
 * event can not be accessed problem.
 */
public void deleteSelectedComponent ()
{
    //prevent null pointer exception 7/28/00 SYT
    if(selectedComponent != null)
    {
        if(selectedComponent.getDataFlowComponent().getChildCount()!=0 )
        {
            //add 2/7/00 SYT
            new DeleteDialog(this,false);
        }
        else
        {
            deleteSelected();
        }
    }
}
// add 2/7/00 SYT
/**
 * delete seleted component and it's children
 */
public void deleteSelected()
{
    DefaultMutableTreeNode selectedParent =
        (DefaultMutableTreeNode)((DefaultMutableTreeNode)
        (selectedComponent.getDataFlowComponent ())).getParent();

```

```

if (!(selectedComponent.getDataFlowComponent () instanceof
    External))
{
    pushUndoVector();
    //delete it and it's childs. SYT
    selectedComponent.delete ();

    displayComponentVector.removeElement (selectedComponent);

    //error. 1/6/00 SYT
    //parentFrame.getTreePanel ().removeDfc (selectedComponent
    //getDataFlowComponent ());
    parentFrame.getTreePanel ().removeDfc();
    // This takes care of unremoved streams
    setParentVertex (parentVertex, null);
    parentFrame.setSaveRequired (true);
}
//change on 2/28/00 SYT
//set path of the selected component
//debug : null pointer exception. 6/12/00 SYT
if(selectedParent != null )
{
    if(selectedParent.isLeaf())
    {
        //add if statement to debug null pointer exception 6/10/00
//SYT
        if( (DefaultMutableTreeNode)selectedParent
            .getParent() != null)
        {
            TreeNode [] obj= ((DefaultMutableTreeNode)selectedParent
                .getParent()).getPath();
            //add 2/12/00 SYT
            //following lines will make labe marker disapear
            TreePath TP = new TreePath(obj);
            parentFrame.getTreePanel().setSelectionPath(TP);
            //ensure selected path visible
            parentFrame.getTreePanel().expandPath(TP);
        }
    }
    else
    {
        TreeNode [] obj= selectedParent.getPath();
        //add 2/12/00 SYT
        //following lines will make label marker disapear
        TreePath TP = new TreePath(obj);
        parentFrame.getTreePanel().setSelectionPath(TP);
        //ensure selected path visible
        parentFrame.getTreePanel().expandPath(TP);
    }
}

//add SYT 12/31/99
/**
 * clear all tree node from root.
 */
public void deleteAllSelectedComponents ()
{
    //add 2/9/00 SYT
    boolean hasChildrenFlag = false;
    //add 2/7/00 SYT
    for(Enumeration e = parentVertex.children() ; e.hasMoreElements()
        ;)
    {
        if( ((DefaultMutableTreeNode)(e.nextElement())).getChildCount()
            != 0)
            hasChildrenFlag=true;
    }
    //add 2/9/00 SYT
    if(hasChildrenFlag)
    {
        new DeleteDialog(this,true);
    }
    else
    {
        deleteAllSelected();
    }
}

```

```

    }
    // add 2/7/00 SYT
    /**
     * for clear all tree nodes
     * and their children from root.
     */
    public void deleteAllSelected ()
    {
        pushUndoVector();
        DataFlowComponent DFC =null;
        if(!handlesVector.isEmpty())
        {
            handlesVector.removeAllElements ();
        }
        if (!displayComponentVector.isEmpty ())
        {
            while (parentVertex.getChildCount() != 0)
            {
                ((DataFlowComponent)(parentVertex.getChildAt(0)) ).delete();
            }

            displayComponentVector.removeAllElements();
        }
        selectAllMode=false;
        //add 1/13/00 SYT
        setSelectMode (true);
        parentFrame.setSaveRequired (true);
        clearAllComponentsFromScreen(null);

        parentFrame.getTreePanel ().removeDfc();
        // setParentVertex (parentVertex, null);
        paint (getGraphics ());
    }

    public Vertex getParentVertex ()
    {
        return parentVertex;
    }

    public void setSelectAllMode (boolean b)
    {
        selectAllMode = b;
        if (selectAllMode)
        {
            selectAllComponents ();
            //add 1/13/00 SYT
            setSelectMode(true);
        }
    }

    //add SYT 12/31/11
    /**
     * current select mode
     */
    public boolean getSelectAllMode()
    {
        return selectAllMode;
    }

    public void selectAllComponents ()
    {
        DisplayComponent dc = null;
        handlesVector.removeAllElements ();
        if (!displayComponentVector.isEmpty ())
        {
            bounds = (Rectangle) ((DisplayComponent)
            displayComponentVector.elementAt (0)).getShape ().getBounds ();
        }
        for (Enumeration enum = displayComponentVector.elements ();
        enum.hasMoreElements ());
        {
            dc = (DisplayComponent) enum.nextElement ();
            handlesVector.addAll (dc.getHandles ());
            handlesVector.addAll (dc.getStringHandles (dc.getLabelShapeBounds ()));
            handlesVector.addAll (dc.getStringHandles (dc.getMetShapeBounds ()));
            bounds = (Rectangle) bounds.createUnion (dc.getShape ().getBounds ());
        }
    }

```

```

    }
    selectedComponent = dc;
    paint (getGraphics ());
}

public void setSelectionDefault (boolean b)
{
    selectionDefault = b;
}

public void setCurrentColor (int colorIndex)
{
    currentColor = colorIndex;
}

public void setCurrentFont (int fontIndex)
{
    currentFont = fontIndex;
}

protected void rubberBandLine (int x, int y)
{
    Point last = (Point) ((Edge) currentEdge.getDataFlowComponent ()).getPoints
().lastElement ();
    Graphics g = getGraphics ();
    g.setColor (new Color (128, 128, 128));
    g.setXORMode (Color.white);
    g.drawLine (last.x, last.x, x, y);
    g.setPaintMode ();
    g.setColor (Color.black);
}
//add ket evet 2/3/00 SYT
public void keyPressed(KeyEvent e)
{
    if(e.getKeyChar()==e.VK_ESCAPE)
    {
        clearnDrawStream();
    }
}

public void keyTyped(KeyEvent e)
{
}

public void keyReleased(KeyEvent e)
{
}

} // End of the class DrawPanel.
/**
 * setting edge's property
 * @author Shen-Yi Tao
 * @version 1.1
 */
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import caps.Psdl.*;
import caps.Display.EdgePath;
import caps.Parser.*;
import javax.swing.event.*;
import java.util.Enumeration;
import javax.swing.text.*;
import javax.swing.tree.DefaultMutableTreeNode;

public class EdgeProperties extends JDialog implements ActionListener
{
    // used to prevent the Java user interface event exception 5/18/00 SYT
    static boolean firstEnter ;

    Edge targetEdge;

    EdgePath ePath;

```

```

JTextField nameField;
JTextField streamTypeField;
JTextField latencyField;
JTextField initValueField;

//create listener 5/18/00 SYT
MyDocumentListener nameFieldListener
    = new MyDocumentListener(nameField);
MyDocumentListener streamTypeFieldListener
    = new MyDocumentListener(streamTypeField);
MyDocumentListener latencyFieldListener
    = new MyDocumentListener(latencyField);
MyDocumentListener initValueFieldListener
    = new MyDocumentListener(initValueField);

JRadioButton noButton;
JRadioButton yesButton;

JComboBox latencyUnitsCombo;

JButton okButton;
JButton cancelButton;
JButton helpButton;
JButton initValueButton;

Editor parentFrame;

public EdgeProperties (Editor parent)
{
    super (parent, "StreamProperties", true);
    parentFrame = parent;
    setResizable (false);
    initialize ();
    pack ();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation ((screenSize.width - getWidth ()) / 2,
        (screenSize.height - getHeight ()) / 2);
}

public void initialize ()
{
    firstEnter = true;

    Box box = Box.createVerticalBox ();

    GridBagConstraints gbc = new GridBagConstraints ();
    gbc.fill = GridBagConstraints.BOTH;
    gbc.insets = new Insets (5, 3, 5, 3);

    JPanel namePanel = new JPanel (new GridBagLayout ());
    namePanel.setBorder (BorderFactory.createTitledBorder (""));
    nameField = new JTextField (15);
    streamTypeField = new JTextField (15);
    noButton = new JRadioButton ("No", true);
    noButton.addActionListener (this);
    yesButton = new JRadioButton ("Yes", false);
    yesButton.addActionListener (this);
    ButtonGroup group = new ButtonGroup ();
    group.add (noButton);
    group.add (yesButton);
    initValueButton = new JButton ("State Initial Value");
    initValueButton.addActionListener (this);
    initValueField = new JTextField (15);
    initValueField.setEditable (false);
    latencyField = new JTextField (10);
    latencyUnitsCombo = getUnitsCombo ();
    gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 0;
    namePanel.add (new JLabel ("Stream Name :"), gbc);
    gbc.gridwidth = 3; gbc.gridx = 2;
    namePanel.add (nameField, gbc);
    gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 1;
    namePanel.add (new JLabel ("Stream Type :"), gbc);
    gbc.gridwidth = 3; gbc.gridx = 2;
    namePanel.add (streamTypeField, gbc);
    gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 2;
    namePanel.add (new JLabel ("Is it a state stream?"), gbc);
    gbc.gridwidth = 1; gbc.gridx = 2;

```

```

namePanel.add (noButton, gbc);
gbc.gridx = 3;
namePanel.add (yesButton, gbc);
gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 3;
namePanel.add (initValueButton, gbc);
gbc.gridwidth = 3; gbc.gridx = 2;
namePanel.add (initValueField, gbc);
gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 4;
namePanel.add (new JLabel ("Latency :"), gbc);
gbc.gridx = 2;
namePanel.add (latencyField, gbc);
gbc.gridwidth = 1; gbc.gridx = 4;
namePanel.add (latencyUnitsCombo, gbc);

JPanel okPanel = new JPanel (new FlowLayout ());
okButton = new JButton ("OK");
okButton.addActionListener (this);
cancelButton = new JButton ("Cancel");
cancelButton.addActionListener (this);
helpButton = new JButton ("Help");
helpButton.addActionListener (this);
gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
okPanel.add (okButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 4;
okPanel.add (cancelButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 7;
okPanel.add (helpButton, gbc);

box.add (Box.createVerticalStrut (2));
box.add (namePanel);
box.add (Box.createVerticalStrut (5));
box.add (okPanel);
box.add (Box.createVerticalStrut (3));

getContentPane ().add (box, BorderLayout.CENTER);
//add document listener 5/17/00 SYT
nameField.getDocument()
.addDocumentListener(new MyDocumentListener(nameField));
streamTypeField.getDocument()
.addDocumentListener(new MyDocumentListener(streamTypeField));
latencyField.getDocument()
.addDocumentListener(new MyDocumentListener(latencyField));
initValueField.getDocument()
.addDocumentListener(new MyDocumentListener(initValueField));
}

public void setEdge (Edge e)
{
    targetEdge = e;
    nameField.setText (e.getLabel ());
    streamTypeField.setText (e.getStreamType ());
    //5/19/00 SYT
    if ( e.getInitialValue().length()==0 )
        e.setStateStream(false);
    else
        e.setStateStream(true);

    if (e.isStateStream () == false)
    {
        noButton.setSelected (true);
        initValueButton.setEnabled (false);
        initValueField.setText ("");
    }
    else
    {
        yesButton.setSelected (true);
        initValueButton.setEnabled (true);
        initValueField.setText (targetEdge.getInitialValue ());
    }

    if (e.getMet () != null)
    {
        PSDLTime latency = e.getMet ();
        latencyField.setText (String.valueOf (latency.getTimeValue
        latencyUnitsCombo.setSelectedIndex (latency.getTimeUnits ());
    }
    else

```



```

    {
        latencyField.setText ("");
        // Set default to ms
        latencyUnitsCombo.setSelectedIndex (1);
    }
}

public void setEdgePath (EdgePath e)
{
    ePath = e;
    setVisible (true);
}

public void actionPerformed (ActionEvent e)
{
    boolean exceptionOccurred = false;
    if (e.getSource () == yesButton)
    {
        initValueButton.setEnabled (true);
        initValueButton.doClick ();
    }
    if (e.getSource () == noButton)
    {
        initValueField.setText ("");
        initValueButton.setEnabled (false);
    }
    if (e.getSource () == initValueButton)
    {
        TextEditor.openDialog ("Stream Initial Value", "View or Edit
Stream Initial Value",
                               initValueField.getText (),
GrammarCheck.INITIAL_EXPRESSION, false);
        initValueField.setText (TextEditor.getString ());
    }
    if (e.getSource () == okButton)
    {
        boolean errorStatus = false;
        String str = nameField.getText ();
        //name field 11/9/99 SYT
        if (!GrammarCheck.isValid (str, GrammarCheck.OP_ID))
        {
            showErrorDialog ("Illegal stream name");
            errorStatus = true;
        }
        str = streamTypeField.getText ();
        if (!GrammarCheck.isValid (str, GrammarCheck.TYPE_NAME))
        {
            showErrorDialog ("Illegal stream type name");
            errorStatus = true;
        }
        str = latencyField.getText ();
        if (str.length () != 0)
        {
            // To be able to delete a latency value
            if (!GrammarCheck.isValid (str, GrammarCheck.INTEGER_LITERAL))
            {
                showErrorDialog ("Illegal value for latency field");
                errorStatus = true;
            }
        }
    }
    if (!errorStatus)
    {
        targetEdge.setLabel (nameField.getText ());
        targetEdge.setStreamType (streamTypeField.getText ());

        parentFrame.getDataTypes ().addType (streamTypeField.getText ());
        if (latencyField.getText ().length () != 0)
            targetEdge.setMet (new PSDLTime (Integer.parseInt
(latencyField.getText ()),
                                             latencyUnitsCombo.getSelectedIndex ()));
    }
    else
        targetEdge.setMet (null);

    if (noButton.isSelected ())
    {
        targetEdge.setInitialValue ("");
        targetEdge.setStateStream (false);
    }
}

```

```

    }
    else
    {
        targetEdge.setInitialValue (initValueField.getText ());
        targetEdge.setStateStream (true);
    }

    if (targetEdge.isStateStream ())
    {
        Vertex parent = (Vertex) targetEdge.getParent ();
        ((java.util.Vector) parent.getSpecReqmts ().elementAt
(2)).addElement
    }

    ePath.setLabelShape ((Graphics2D) parentFrame.getDrawPanel ()
        .getGraphics
        ());
    ePath.setMetShape ((Graphics2D) parentFrame.getDrawPanel ()
        .getGraphics
        ());

    setVisible (false);
    parentFrame.getDrawPanel ().clearAllComponentsFromScreen
(null);
    parentFrame.getDrawPanel ()`
        .paint (parentFrame.getDrawPanel
().getGraphics ());
    parentFrame.getTreePanel ().repaint ();
    parentFrame.setSaveRequired (true);

    //8/26/00 check Edge consistence SYT
    StreamConsistenceCheck
        .checkConsistence(
(DefaultMutableTreeNode)(targetEdge.getRoot())
, targetEdge );

    }

    }
    if (e.getSource () == cancelButton)
    {
        setVisible (false);
    }
    if (e.getSource () == helpButton)
    {
        System.out.println ("Help not available now");
    }
}

public void showErrorDialog (String str)
{
    JOptionPane.showMessageDialog (this, str, "Error Message"
        , JOptionPane.ERROR_MESSAGE);
}

public JComboBox getUnitsCombo ()
{
    JComboBox c =new JComboBox ();
    c.addItem ("microsec");
    c.addItem ("ms");
    c.addItem ("sec");
    c.addItem ("min");
    c.addItem ("hours");
    return c;
}
/**
 *inner class to handle document listener 5/17/00 SYT
 */
class MyDocumentListener implements DocumentListener
{
    private JTextField JTF;
    public MyDocumentListener(JTextField t)
    {
        JTF = t;
    }
    public void insertUpdate(DocumentEvent e)

```

```

{
    if(!firstEnter)
    {
        if(JTF == nameField);
        updatedialog(e);
    }
    if(isVisible())
    {
        firstEnter=false;
    }
    else
    {
        firstEnter=true;
    }
}
public void removeUpdate(DocumentEvent e)
{
    if(!firstEnter)
    {
        if(JTF == nameField);
        updatedialog(e);
    }
    if(isVisible())
    {
        firstEnter=false;
    }
    else
    {
        firstEnter=true;
    }
}
public void changedUpdate(DocumentEvent e)
{
}
}
//5/17/00 SYT
/**
 *update edge propriety for the same stream
 */
public void updatedialog(DocumentEvent d)
{
    Document doc = (Document)d.getDocument();

    String str="";
    try
    {
        str = doc.getText(0,doc.getLength());
    }
    catch (BadLocationException b)
    {
    }
    for( Enumeration e = ( (DefaultMutableTreeNode) (targetEdge.getRoot()) )
        .breadthFirstEnumeration(); e.hasMoreElements(); )
    {
        DataFlowComponent DFC = (DataFlowComponent)(e.nextElement());
        if( !DFC.equals(targetEdge) )
        {
            if(DFC instanceof Edge)
            {
                if(str.equals( DFC.getLabel() ) )
                {
                    copyEdge((Edge)DFC);
                }
            }
        }
    }
}
// 5/17/00 SYT
/**
 * update edge propriety for the same stream
 */
public void copyType(Edge d )
{
    String str = d.getLabel() ;

```

```

for( Enumeration e = ( (DefaultMutableTreeNode) (d.getRoot()) )
    .breadthFirstEnumeration(); e.hasMoreElements(); )
{
    DataFlowComponent DFC = (DataFlowComponent)(e.nextElement());
    if( (DFC instanceof Edge) && (!targetEdge.equals(DFC)) )
    {
        if(str.equals(DFC.getLabel()) )
        {
            ((Edge)DFC).setStreamType(streamTypeField.getText());

            ((Edge)DFC).setStateStream (targetEdge.isStateStream());

            if(targetEdge.isStateStream())
                ((Edge)DFC).setInitialValue(initValueField.getText());
            else
                ((Edge)DFC).setInitialValue("");

            PSDLTime met = targetEdge.getMet();
            ((Edge)DFC).setMet (met);

            //((Edge)DFC).setMet (new PSDLTime (Integer.parseInt (
            //latencyField.getText ()),latencyUnitsCombo.getSelectedIndex ());
        }
    }
}

public void copyEdge (Edge e)
{
    streamTypeField.setText (e.getStreamType ());
    if (e.isStateStream () == false)
    {
        noButton.setSelected (true);
        initValueButton.setEnabled (false);
        initValueField.setText ("");
    }
    else
    {
        yesButton.setSelected (true);
        initValueButton.setEnabled (true);
        initValueField.setText (e.getInitialValue ());
    }

    if (e.getMet () != null)
    {
        PSDLTime latency = e.getMet ();
        latencyField.setText (String.valueOf (latency.getTimeValue ()));
        latencyUnitsCombo.setSelectedIndex (latency.getTimeUnits ());
    }
    else
    {
        latencyField.setText ("");
        latencyUnitsCombo.setSelectedIndex (1);    // Set default to ms
    }
}

} // End of the class EdgeProperties

package caps.GraphEditor;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import caps.Psdl.*;
import caps.Psdl.DataTypes;
import caps.Parser.*;
import java.io.*;
import caps.Ada.AdaTemplet;
import java.util.Enumeration;

/**
 * The main frame for the Graph Editor.
 * It constructs and drives the other features.
 * @author Shen-Yi Tao
 * @version 1.1
 */

```

```

//add key listener,mouse listener here
//SYT 12/30/99
public class Editor extends JFrame implements Runnable ,KeyListener
{
//public class Editor extends JFrame implements Runnable {

    /**
     * The panel that includes the Drawing area and tree view
     */
    protected JPanel panel;

    /**
     * Includes the treePanel and the drawPanel.
     */
    protected JSplitPane innerSplit;

    /**
     * The panel that includes the tree structure to view
     */
    protected TreePanel treePanel;

    /**
     * The panel that the drawing operations are performed.
     */
    protected DrawPanel drawPanel;

    protected StatusBar statusBar;

    /**
     * the main toolbar of the GraphEditor
     */
    protected ToolBar tBar;

    // root of this vertex tree. SYT 12/30/99
    protected Vertex root;

    /**
     * The initial width of the GraphEditor
     */
    private final int INITIAL_WIDTH = 800;

    /**
     * The initial height of the Graph Editor
     */
    private final int INITIAL_HEIGHT = 600;

    protected DataTypes types;

    protected File prototypeFile;
    protected File adaTempletFolderName;

    protected boolean saveRequired;

    //add 1/2/00 SYT
    protected EditorMenuBar aEditorMenuBar;

    // 11/16/99 SYT
    /**
     * getting phototypeName.
     */
    public static String prototypeName;
    /**
     * counter for how many ada files
     */
    private int counter = 0;

    /**
     * The constructor for the editor frame
     */
    public Editor (File prototype, File adaTemplate, Vertex r, DataTypes
t)
    {
        //super ("PSDL Editor");

        // 11/16/99 initialize prototypeName. SYT
        prototypeName = prototype.getName();

```

```

        prototypeFile = prototype;

        adaTempletFolderName = adaTemplate;

        root = r;

        saveRequired = false;

        types = t;
        //add key listener. SYT 12/30/99
        addKeyListener(this);
        //initialize ();
    }

    // this is another thread to paint main window
    public void run ()
    {
        initialize ();
    }

    /**
     * The initialization of the GUI takes place here
     */
    public void initialize ()
    {
        // Set the look and feel to a platform independent view
        try
        {
            UIManager.setLookAndFeel (UIManager
                                     .getCrossPlatformLookAndFeelClassName
                                     ());
        } catch (Exception e)
        {
            System.err.println("Error loading L&F: " + e);
        }

        setTitle ("PSDL Editor : " + prototypeFile.getName ());
        setDefaultCloseOperation (WindowConstants.DO_NOTHING_ON_CLOSE);
        addWindowListener (new ExitEditor (this));

        //add SYT 1/2/00
        aEditorMenuBar = new EditorMenuBar (this);

        statusBar = new StatusBar (this);

        //SYT 1/2/00
        //setJMenuBar (new EditorMenuBar (this));
        setJMenuBar(aEditorMenuBar);
        setSize (INITIAL_WIDTH, INITIAL_HEIGHT);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation((screenSize.width - INITIAL_WIDTH) / 2,
                    (screenSize.height - INITIAL_HEIGHT) / 2);

        BorderLayout bLayout = new BorderLayout ();
        bLayout.setVgap (3);

        panel = new JPanel (bLayout);
        panel.setAlignmentX (LEFT_ALIGNMENT);
        panel.setAlignmentY (TOP_ALIGNMENT);
        panel.setBorder (BorderFactory.createLoweredBevelBorder ());

        getContentPane ().add (panel);

        tBar = new ToolBar (this);

        root.setAllowsChildren (true);
        // root is the root of the tree.
        // create Tree panel. SYT 12/30/99
        treePanel = new TreePanel (this, root);
        // root is the root of the vertex tree.
        // create Drawpanel. SYT 12/30/99
        drawPanel = new DrawPanel (this, root);

        JScrollPane p1 = new JScrollPane (treePanel);
        p1.setBackground (Color.white);
    }

```

```

        JScrollPane p2 = new JScrollPane (drawPanel);
        p2.setBackground (Color.white);

        innerSplit = new JSplitPane (JSplitPane.HORIZONTAL_SPLIT, p1, p2);
        innerSplit.setContinuousLayout(true);
        innerSplit.setOneTouchExpandable (true);
        innerSplit.setDividerLocation (getWidth () / 5);
        innerSplit.setBorder (BorderFactory.createLoweredBevelBorder ());

        panel.add (tBar, BorderLayout.NORTH);
        panel.add (innerSplit, BorderLayout.CENTER);
        panel.add (statusBar, BorderLayout.SOUTH);

        //add to test. SYT
        //PsdParser.enable_tracing();

        // Set this to true if you want to trace the parser actions
        PsdParser.disable_tracing();

        setVisible (true);

        drawPanel.setParentVertex (root, null);
    }

    public EditorMenuBar getEditorMenuBar()
    {
        return aEditorMenuBar;
    }

    /**
     * Returns the TreePanel object in this frame
     *
     * @return the treePanel object in this JFrame
     */
    public TreePanel getTreePanel ()
    {
        return treePanel;
    }

    /**
     * Returns the DrawPanel object in this frame
     *
     * @return the drawPanel object in this JFrame
     */
    public DrawPanel getDrawPanel ()
    {
        return drawPanel;
    }

    /**
     * Returns the toolBar object in this frame
     *
     * @return the toolBar object in this JFrame
     */
    public ToolBar getToolBar ()
    {
        return tBar;
    }

    public StatusBar getStatusBar ()
    {
        return statusBar;
    }

    public JSplitPane getSplitPane ()
    {
        return innerSplit;
    }

    public Vertex getRoot ()
    {
        return root;
    }

    public DataTypes getDataTypes ()
    {
        return types;
    }

```

```

    }

    public File getPrototypeFile ()
    {
        return prototypeFile;
    }

    public void setSaveRequired (boolean b)
    {
        saveRequired = b;
        if (saveRequired)
        {
            statusBar.setText ("Save required");
            // enable save menu item
            getJMenuBar ().getMenu (0).getItem (0).setEnabled (true);
        }
        else
        {
            statusBar.setText ("Save not required");
            // disable save menu item
            getJMenuBar ().getMenu (0).getItem (0).setEnabled (false);
        }
    }

    public boolean isSaveRequired ()
    {
        return saveRequired;
    }

    public boolean checkSaved ()
    {
        boolean value = true;
        if (saveRequired)
        {
            int ix = JOptionPane.showConfirmDialog (this
                , new String ("Save changes to the prototype?"));
            if (ix == JOptionPane.CANCEL_OPTION)
                value = false;
            else if (ix == JOptionPane.YES_OPTION)
                savePrototype ();
        }
        return value;
    }
    //modified 8/25/00
    /**
     * save both prototype and ada template to files
     */
    public void savePrototype ()
    {
        try
        {
            setCursor (Cursor.WAIT_CURSOR);
            FileWriter testFile = new FileWriter(prototypeFile);
            //add 8/26/00 SYT
            for(Enumeration e = root.breadthFirstEnumeration();
            e.hasMoreElements();)
            {
                DataFlowComponent DFC =((DataFlowComponent)
                e.nextElement());
                if ( DFC instanceof Vertex & !DFC.isRoot())
                    counter++;
            }

            FileWriter [] adaFile = new FileWriter [counter];

            if (!adaTempletFolderName.exists() )
                adaTempletFolderName.mkdir ();

            // else
            //  adaTempletFolderName.

            int row =0;
            for(Enumeration e = root.breadthFirstEnumeration();
            e.hasMoreElements();)
            {

```



```

        DataFlowComponent DFC = ((DataFlowComponent)
e.nextElement());
        if ( DFC instanceof Vertex & !DFC.isRoot())
        {
            if( ((Vertex)DFC).isLeaf() &
((Vertex)DFC).getImpLanguage().equalsIgnoreCase("ada") )
            {
                File temp = new File(adaTempletFolderName.toString()
+ File.separator + DFC.getLabel() + "_" +
Integer.toString(DFC.getId())+".at" );
                adaFile[row] = new FileWriter(temp);
                row++;
            }
        }
    }

StringWriter writer = new StringWriter ();
CreatePsdل.ReInit (writer);

CreatePsdل.build (root, types);
//get PSDL 11/3/99 SYT
String str = CreatePsdل.getPsdل ();

//check structure 6/10/00 SYT
StructureCheck.psdلStructureCheck(root);

//check the global edge consistence 8/27/00 SYT
StreamConsistenceCheck.checkConsistence(root,null);

//check grammar 11/5/99 SYT
PsdلParser.ReInit (new StringReader (str));
PsdلParser.psdل ();
testFile.write (str);
testFile.close ();

//create Ada templet 8/26/00 SYT
AdaTemplet Templet = new AdaTemplet(root);
Templet.writeTemplet();

StringWriter [] templetStr = Templet.getTemplet();

if (!adaTempletFolderName.exists() )
    adaTempletFolderName.mkdir ();

for(int i = 0 ; i<counter ;i++)
{
    if( adaFile[i] != null )
    {
        adaFile[i].write(templetStr[i].toString());

        adaFile[i].close();

        adaFile[i] = null;
    }
}

//PsdلParser.ReInit (new StringReader (str));
//PsdلParser.psdل ();
writer.close ();
setCursor (Cursor.DEFAULT_CURSOR);
setSaveRequired (false);

}
catch (IOException ex)
{
    System.out.println (ex);
}
catch (ParseException ex)
{
    //System.out.println (ex);
    showErrorDialog ("Parse exception encountered " + ex);
} //catch (Exception ex)
//{
//    System.out.println (ex);
//}
}

```

```

public void showErrorDialog (String str)
{
    JOptionPane.showMessageDialog
        (this, str, "Error Message", JOptionPane.ERROR_MESSAGE);
}

// add 12/30/99 SYT
public void keyPressed(KeyEvent e)
{
    if(e.getKeyChar() == e.VK_DELETE)
    {
        if(!drawPanel.getSelectedAllMode() )
            drawPanel.deleteSelectedComponent();
        else
            drawPanel.deleteAllSelectedComponents();
    }
}

public void keyTyped(KeyEvent e)
{
}

// add. SYT 12/30/99
public void keyReleased(KeyEvent e)
{
    if(e.getKeyChar() == e.VK_ESCAPE)
    {
        drawPanel.clearnDrawStream();
    }
}

} // End of the class Editor
package caps.GraphEditor;

import javax.swing.*;

/**
 * The MenuBar of the Graph Editor.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class EditorMenuBar extends JMenuBar
{
    //add 1/2/99 SYT
    private GE_FileMenu aFileMenu;
    private GE_EditMenu aEditMenu;
    private GE_ViewMenu aViewMenu;
    private GE_PSDLMenu aPSDLMenu;
    private GE_HelpMenu aHelpMenu;

    /**
     * The constructor for this class.
     */
    public EditorMenuBar (Editor parent)
    {
        super ();
        //add SYT 1/2/00
        aFileMenu = new GE_FileMenu (parent);
        aEditMenu = new GE_EditMenu (parent);
        aViewMenu = new GE_ViewMenu (parent);
        aPSDLMenu = new GE_PSDLMenu (parent);
        aHelpMenu = new GE_HelpMenu (parent);
        //modified 1/2/00 SYT
        add (aFileMenu);
        add (aEditMenu);
        add (aViewMenu);
        add (aPSDLMenu);
        add (aHelpMenu);
    }

    //you can create more get_menu_fucntions, currently they are not needed.
    //SYT 1/2/00
    public GE_EditMenu getEditMenu()
    {
        return aEditMenu;
    }
}

} // End of the class EditorMenuBar

```

```

package caps.GraphEditor;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/**
 * Closes the caps main window and exits from the program.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
class ExitEditor extends WindowAdapter
{
    Editor editor;

    public ExitEditor (Editor e)
    {
        editor = e;
    }

    /**
     * Window event handler for the menu events.
     *
     * @param e The window event that is created when the program
     * close icon is pressed.
     */
    public void windowClosing (WindowEvent e)
    {
        if (editor.checkSaved ()) {
            caps.CAPSMMain.CAPSMMainWindow.removeEditor (editor);
            editor.setVisible (false);
            editor.dispose ();
        }
        //System.exit (0);
    }
} // End of the class ExitEditor
package caps.GraphEditor;

public class FontConstants {

    public static String FONT_VALUES [] = { "Courier", "Bold", "10", "Courier"
        , "Bold", "12", "Courier", "Bold", "14", "Courier", "Plain", "10", "Courier"
        , "Plain", "12", "Courier", "Plain", "14" };

    public static String FONT_NAMES [] = { "Courier Bold 10", "Courier Bold 12"
        , "Courier Bold 14"
        , "Courier Plain 10", "Courier Plain 12", "Courier Plain 14" };

} // End of the class ColorConstants
package caps.GraphEditor;

import javax.swing.*.*;
import java.awt.event.*.*;

/**
 * Constructs the Edit menu of the menubar.
 * Also handles the events associated with the Edit Menu.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class GE_EditMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'Undo' event
     */
    private JMenuItem undoMenuItem = new JMenuItem ("Undo");

    /**
     * Initiates the 'Redo' event
     */
    private JMenuItem redoMenuItem = new JMenuItem ("Redo");

    /**
     * Initiates the 'Copy' event
     */

```

```

//private JMenuItem copyMenuItem = new JMenuItem ("Copy");

/**
 * Initiates the 'Paste' event
 */
//private JMenuItem pasteMenuItem = new JMenuItem ("Paste");

private JMenuItem selectAllMenuItem = new JMenuItem ("Select All");

/**
 * Initiates the 'Delete' event
 */
private JMenuItem deleteMenuItem = new JMenuItem ("Delete");

private Editor parent;

/**
 * The constructor for the Edit menu
 */
public GE_EditMenu (Editor e)
{
    super ("Edit");

    parent = e;

    add (undoMenuItem);
    add (redoMenuItem);
    addSeparator ();
    //add (copyMenuItem);
    //add (pasteMenuItem);
    add (selectAllMenuItem);
    add (deleteMenuItem);

    /*
     * These are not implemented yet
     * Take these lines out when they are implemented
     */
    undoMenuItem.setEnabled (false);
    redoMenuItem.setEnabled (false);
    deleteMenuItem.setEnabled (false);
    //copyMenuItem.setEnabled (false);
    //pasteMenuItem.setEnabled (false);

    undoMenuItem.setActionCommand ("Undo last action");
    redoMenuItem.setActionCommand ("Redo last action");
    //copyMenuItem.setActionCommand
    //      ("Copy selected component into clipboard");
    //pasteMenuItem.setActionCommand
    //      ("Paste the component in the clipboard into the      //drawing
area");
    selectAllMenuItem
        .setActionCommand ("Selects all the components on the drawing
area");
    deleteMenuItem.setActionCommand ("Delete the selected      component");

    undoMenuItem.addMouseListener (e.getStatusBar ());
    redoMenuItem.addMouseListener (e.getStatusBar ());
    selectAllMenuItem.addMouseListener (e.getStatusBar ());
    //copyMenuItem.addMouseListener (e.getStatusBar ());
    //pasteMenuItem.addMouseListener (e.getStatusBar ());
    deleteMenuItem.addMouseListener (e.getStatusBar ());

    undoMenuItem.addActionListener (this);
    redoMenuItem.addActionListener (this);
    selectAllMenuItem.addActionListener (this);
    //copyMenuItem.addActionListener (this);
    //pasteMenuItem.addActionListener (this);
    deleteMenuItem.addActionListener (this);
}

/**
 * Handles the menu events that occur when one of the menu items
 * is selected
 * @param e The associated ActionEvent
 */
public void actionPerformed (ActionEvent e)
{

```

```

        if (e.getSource () == undoMenuItem)
        {
            parent.getDrawPanel().undoPaint();
            //add 7/25/00 SYT
            parent.setSaveRequired(true);
        }
        else if (e.getSource () == redoMenuItem)
        {
            //add 1/2/99 SYT
            parent.getDrawPanel().redoPaint();
            //add 7/25/00 SYT
            parent.setSaveRequired(true);
        }

        else if (e.getSource () == selectAllMenuItem)
        {
            parent.getDrawPanel ().setSelectAllMode (true);
            //add. SYT 12/31/99
            deleteMenuItem.setEnabled (true);
        }
        // SYT 12/30/99
        //else if (e.getSource () == deleteMenuItem) {
        // parent.getDrawPanel ().deleteAllSelectedComponents();
        else if (e.getSource () == deleteMenuItem)
        {
            if(!parent.getDrawPanel().getSelectAllMode() )
                parent.getDrawPanel().deleteSelectedComponent();
            //undoMenuItem.setEnabled(true);
            else
                parent.getDrawPanel().deleteAllSelectedComponents();
        }
    }
    // add SYT 1/2/00
    /**
     * get undo menu item
     */
    public JMenuItem getUndoMenuItem()
    {
        return undoMenuItem;
    }
    public JMenuItem getRedoMenuItem()
    {
        return redoMenuItem;
    }
} // End of the class GE_EditMenu

package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;
import java.awt.Cursor;
import caps.Psdl.Vertex;

/**
 * Constructs the File menu of the menubar.
 * Also handles the events associated with the File Menu.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class GE_FileMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'Save' event
     */
    private JMenuItem saveMenuItem = new JMenuItem ("Save");

    /**
     * Initiates the 'Restore From Save' event
     */
    private JMenuItem restoreMenuItem = new JMenuItem ("Restore From Save");

    /**
     * Initiates the 'Print' event
     */
    private JMenuItem printMenuItem = new JMenuItem ("Print");

```

```

/**
 * Initiates the 'Exit' event
 */
private JMenuItem exitMenuItem = new JMenuItem ("Exit");

private Editor parent;

/**
 * The constructor for the File menu
 */
public GE_FileMenu (Editor e)
{
    super ("File");

    parent = e;

    add (saveMenuItem);
    add (restoreMenuItem);
    add (printMenuItem);
    add (exitMenuItem);

    /**
     * These are not implemented yet
     * Take these lines out when they are implemented
     */
    restoreMenuItem.setEnabled (false);

    saveMenuItem.setEnabled (false);

    saveMenuItem.setActionCommand ("Save the prototype into disk");
    restoreMenuItem.setActionCommand ("Restore saved prototype from disk");
    printMenuItem.setActionCommand ("Print the prototype");
    exitMenuItem.setActionCommand ("Quit the graph editor");

    saveMenuItem.addMouseListener (e.getStatusBar ());
    restoreMenuItem.addMouseListener (e.getStatusBar ());
    printMenuItem.addMouseListener (e.getStatusBar ());
    exitMenuItem.addMouseListener (e.getStatusBar ());

    saveMenuItem.addActionListener (this);
    restoreMenuItem.addActionListener (this);
    printMenuItem.addActionListener (this);
    exitMenuItem.addActionListener (this);
}

/**
 * Handles the menu events that occur when one of the menu items
 * is selected
 *
 * @param e The associated ActionEvent
 */
public void actionPerformed (ActionEvent e)
{
    /**
     * 11/3/99 SYT
     * This does not save file during some condition.
     * Go to check out Editor.
     */
    if (e.getSource () == saveMenuItem)
    {
        parent.savePrototype ();
    }
    else if (e.getSource () == restoreMenuItem)
    {
        System.out.println ("Restore has not yet been implemented");
    }
    else if (e.getSource () == printMenuItem)
    {
        DrawPanel panel = parent.getDrawPanel ();
        PrintJob.print (panel, (Vertex) panel.getParentVertex ().getRoot ());
    }
    else if (e.getSource () == exitMenuItem)
    {
        if (parent.checkSaved ())
        {
            caps.CAPSMMain.CAPSMMainWindow.removeEditor (parent);
        }
    }
}

```

```

        parent.dispose ();
        parent.setVisible (false);
    }
}

} // End of the class GE_FileMenu
package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;

/**
 * Constructs the Help menu of the menubar.
 * Also handles the events associated with the Help Menu.
 *
 * @version 1.0
 */
public class GE_HelpMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'PSDL Grammar' event
     */
    private JMenuItem psdlGrammarMenuItem = new JMenuItem ("PSDL Grammar");

    /**
     * Initiates the 'Operators' event
     */
    private JMenuItem operatorsMenuItem = new JMenuItem ("Operators");

    /**
     * Initiates the 'Streams' event
     */
    private JMenuItem streamsMenuItem = new JMenuItem ("Streams");

    /**
     * Initiates the 'Exceptions' event
     */
    private JMenuItem exceptionsMenuItem = new JMenuItem ("Exceptions");

    /**
     * Initiates the 'Timers' event
     */
    private JMenuItem timersMenuItem = new JMenuItem ("Timers");

    /**
     * The constructor for the Help menu
     */
    public GE_HelpMenu (Editor e)
    {
        super ("Help");

        add (psdlGrammarMenuItem);
        add (operatorsMenuItem);
        add (streamsMenuItem);
        add (exceptionsMenuItem);
        add (timersMenuItem);

        /*
         * These are not implemented yet
         * Take these lines out when they are implemented
         */
        psdlGrammarMenuItem.setEnabled (false);
        operatorsMenuItem.setEnabled (false);
        streamsMenuItem.setEnabled (false);
        exceptionsMenuItem.setEnabled (false);
        timersMenuItem.setEnabled (false);

        psdlGrammarMenuItem.setActionCommand ("Opens help about PSDL Grammar");
        operatorsMenuItem.setActionCommand ("Opens help about operators");
        streamsMenuItem.setActionCommand ("Opens help about streams");
        exceptionsMenuItem.setActionCommand ("Opens help about exceptions");
        timersMenuItem.setActionCommand ("Opens help about timers");

        psdlGrammarMenuItem.addMouseListener (e.getStatusBar ());
        operatorsMenuItem.addMouseListener (e.getStatusBar ());
        streamsMenuItem.addMouseListener (e.getStatusBar ());
    }
}

```

```

        exceptionsMenuItem.addMouseListener (e.getStatusBar ());
        timersMenuItem.addMouseListener (e.getStatusBar ());

        psdlGrammarMenuItem.addActionListener (this);
        operatorsMenuItem.addActionListener (this);
        streamsMenuItem.addActionListener (this);
        exceptionsMenuItem.addActionListener (this);
        timersMenuItem.addActionListener (this);
    }

    /**
     * Handles the menu events that occur when one of the menu items
     * is selected
     *
     * @param e The associated ActionEvent
     */
    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == psdlGrammarMenuItem)
        {
            System.out.println ("PSDL Grammar help has not yet been      implemented");
        }
        else if (e.getSource () == operatorsMenuItem)
        {
            System.out.println ("Operators help has not yet been      implemented");
        }
        else if (e.getSource () == streamsMenuItem)
        {
            System.out.println ("Streams help has not yet been      implemented");
        }
        else if (e.getSource () == exceptionsMenuItem)
        {
            System.out.println ("Exceptions help has not yet been      implemented");
        }
        else if (e.getSource () == timersMenuItem)
        {
            System.out.println ("Timers help has not yet been      implemented");
        }
    }

} // End of the class GE_HelpMenu
package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;

/**
 * Constructs the PSDL menu of the menubar.
 * Also handles the events associated with the PSDL Menu.
 *
 * @version 1.0
 */
public class GE_PSDLMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'Goto Root' event
     */
    private JMenuItem gotoRootMenuItem = new JMenuItem ("Goto Root");

    /**
     * Initiates the 'Goto Parent' event
     */
    // private JMenuItem gotoParentMenuItem = new JMenuItem ("Goto      //Parent");

    /**
     * Initiates the 'Decompose' event
     */
    private JMenuItem decomposeMenuItem = new JMenuItem ("Decompose");

    private Editor parent;

    /**
     * The constructor for the PSDL menu
     */
    public GE_PSDLMenu (Editor e)
    {
        super ("PSDL");
    }

```



```

        parent = e;

        add (gotoRootMenuItem);
        //add (gotoParentMenuItem);
        add (decomposeMenuItem);

        decomposeMenuItem.setEnabled (false);
        //gotoParentMenuItem.setEnabled (false);

        gotoRootMenuItem.setActionCommand ("Goto the root operator");
        //gotoParentMenuItem.setActionCommand ("Goto the parent vertex");
        decomposeMenuItem.setActionCommand ("Decompose the selected component");

        gotoRootMenuItem.addMouseListener (parent.getStatusBar ());
        //gotoParentMenuItem.addMouseListener (parent.getStatusBar ());
        decomposeMenuItem.addMouseListener (parent.getStatusBar ());

        gotoRootMenuItem.addActionListener (this);
        //gotoParentMenuItem.addActionListener (this);
        decomposeMenuItem.addActionListener (this);
    }

    /**
     * Handles the menu events that occur when one of the menu items
     * is selected
     * @param e The associated ActionEvent
     */
    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == gotoRootMenuItem)
        {
            parent.getDrawPanel ().gotoRoot ();
        }
        /*else if (e.getSource () == gotoParentMenuItem)
        {
            parent.getDrawPanel ().gotoParent ();
        }*/
        else if (e.getSource () == decomposeMenuItem)
        {
            parent.getDrawPanel ().decompose ();
        }
    }

} // End of the class GE_PSDLMenu
package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;

/**
 * Constructs the View menu of the menubar.
 * Also handles the events associated with the View Menu.
 *
 * @version 1.0
 */
public class GE_ViewMenu extends JMenu implements ActionListener
{
    /**
     * Initiates the 'Color' event
     */
    private JMenuItem colorMenuItem = new JMenuItem ("Color");

    /**
     * Initiates the 'Font' event
     */
    private JMenuItem fontMenuItem = new JMenuItem ("Font");

    /**
     * Initiates the 'Refresh' event
     */
    private JMenuItem refreshMenuItem = new JMenuItem ("Refresh");

    /**
     * Initiates the 'Tree View' event
     */

```

```

private JCheckBoxMenuItem treeViewMenuItem = new JCheckBoxMenuItem      ("Tree");

private JCheckBoxMenuItem toolTipsMenuItem =
    new JCheckBoxMenuItem ("Tool Tips");

private JCheckBoxMenuItem selectionModeMenuItem =
    new JCheckBoxMenuItem ("Auto Select Mode");

private TooltipManager manager;

private Editor parentFrame;

/**
 * The constructor for the View menu
 */
public GE_ViewMenu (Editor parent)
{
    super ("View");

    parentFrame = parent;

    treeViewMenuItem.setSelected (true);
    toolTipsMenuItem.setSelected (true);
    selectionModeMenuItem.setSelected (false);

    manager = TooltipManager.sharedInstance ();
    manager.setEnabled (true);
    manager.setInitialDelay (400);

    add (colorMenuItem);
    add (fontMenuItem);
    addSeparator ();
    add (treeViewMenuItem);
    add (toolTipsMenuItem);
    add (selectionModeMenuItem);
    addSeparator ();
    add (refreshMenuItem);

    /*
     * These are not implemented yet
     * Take these lines out when they are implemented
     */
    //colorMenuItem.setEnabled (false);
    //fontMenuItem.setEnabled (false);

    colorMenuItem.setActionCommand("Changes the current color of the      editor");
    fontMenuItem.setActionCommand ("Changes the current font of the      editor");
    treeViewMenuItem.setActionCommand ("Makes visible/hides the tree      view");
    toolTipsMenuItem.setActionCommand ("Enables/Disables tooltips");
    refreshMenuItem
        .setActionCommand ("Refreshes the display on the drawing      area");
    selectionModeMenuItem
        .setActionCommand ("Defaults to select mode after each
insertion"
        + " of a component on the drawing      area");

    colorMenuItem.addMouseListener (parentFrame.getStatusBar ());
    fontMenuItem.addMouseListener (parentFrame.getStatusBar ());
    treeViewMenuItem.addMouseListener (parentFrame.getStatusBar ());
    toolTipsMenuItem.addMouseListener (parentFrame.getStatusBar ());
    refreshMenuItem.addMouseListener (parentFrame.getStatusBar ());
    selectionModeMenuItem.addMouseListener (parentFrame.getStatusBar      ());

    colorMenuItem.addActionListener (this);
    fontMenuItem.addActionListener (this);
    treeViewMenuItem.addActionListener (this);
    toolTipsMenuItem.addActionListener (this);
    refreshMenuItem.addActionListener (this);
    selectionModeMenuItem.addActionListener (this);
}

/**
 * Handles the menu events that occur when one of the menu items
 * is selected
 *
 * @param e The associated ActionEvent
 */

```

```

public void actionPerformed (ActionEvent e)
{
    if (e.getSource () == colorMenuItem)
    {
        String selected = (String) JOptionPane.showInputDialog
            (parentFrame, "Select color : ", "Color Selection"
            , JOptionPane.INFORMATION_MESSAGE, null
            , ColorConstants.COLOR_NAMES
            , ColorConstants.COLOR_NAMES [0]);
        if (selected != null)
        {
            int colorIndex = 0;
            for (int ix = 0; ix < ColorConstants.COLOR_NAMES.length; ix++)
            {
                if (ColorConstants.COLOR_NAMES [ix].equals (selected))
                    colorIndex = ix;
            }
            parentFrame.getDrawPanel ().setCurrentColor (colorIndex);
        }
    }
    else if (e.getSource () == fontMenuItem)
    {
        String selected = (String) JOptionPane
            .showInputDialog (parentFrame, "Select Font : ", "Font Selection"
            , JOptionPane.INFORMATION_MESSAGE, null, FontConstants.FONT_NAMES
            , FontConstants.FONT_NAMES [0]);
        if (selected != null)
        {
            int fontIndex = 0;
            for (int ix = 0; ix < FontConstants.FONT_NAMES.length; ix++)
            {
                if (FontConstants.FONT_NAMES [ix].equals (selected))
                    fontIndex = ix;
            }
            parentFrame.getDrawPanel ().setCurrentFont (fontIndex);
        }
    }
    else if (e.getSource () == treeViewMenuItem)
    {
        if (!treeViewMenuItem.isSelected ())
            parentFrame.getSplitPane ().setDividerLocation (0.0);
        else
            parentFrame.getSplitPane ().setDividerLocation (
                parentFrame.getSplitPane
                    ().getLastDividerLocation ());
    }
    else if (e.getSource () == toolTipsMenuItem)
    {
        if (toolTipsMenuItem.isSelected ())
            manager.setEnabled (true);
        else
            manager.setEnabled (false);
    }
    else if (e.getSource () == refreshMenuItem)
    {
        DrawPanel panel = parentFrame.getDrawPanel ();
        panel.paint (panel.getGraphics ());
    }
    else if (e.getSource () == selectionModeMenuItem)
    {
        if (selectionModeMenuItem.isSelected ())
            parentFrame.getDrawPanel ().setSelectionDefault (true);
        else
            parentFrame.getDrawPanel ().setSelectionDefault (false);
    }
}

} // End of the class GE_ViewMenu
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import caps.Parser.GrammarCheck;
import java.util.*;

public class IdListEditor implements ActionListener
{

```

```

private static JDialog dialog;

private static JPanel south;

private static final int WIDTH = 400;

private static final int HEIGHT = 300;

protected static Vector vector;

protected static JButton okButton;
protected static JButton cancelButton;
protected static JButton helpButton;
protected static JButton addButton;
protected static JButton deleteButton;
protected static JButton editButton;

protected static JList inputArea;

protected static DefaultListModel model;

protected static JLabel promptLabel;

protected Editor parentFrame;

public IdListEditor (Editor parent)
{
    parentFrame = parent;
    dialog = new JDialog (parentFrame, true);
    dialog.setSize (WIDTH, HEIGHT);
    dialog.setResizable (false);
    dialog.setVisible (false);
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    dialog.setLocation ((screenSize.width - dialog.getWidth ()) / 2,
        (screenSize.height - dialog.getHeight ()) / 2);
    initialize ();
}

protected void initialize ()
{
    dialog.getContentPane ().setLayout (new BorderLayout(5, 5));

    south = new JPanel ();
    dialog.getContentPane ().add (south, BorderLayout.SOUTH);

    promptLabel = new JLabel ();
    dialog.getContentPane ().add (promptLabel, BorderLayout.NORTH);

    model = new DefaultListModel ();
    inputArea = new JList (model);
    inputArea.setBorder (BorderFactory.createLoweredBevelBorder ());
    JScrollPane p = new JScrollPane (inputArea);
    p.setBackground (Color.lightGray);
    dialog.getContentPane ().add (p, BorderLayout.CENTER);

    okButton = new JButton ("OK");
    okButton.addActionListener (this);
    cancelButton = new JButton ("Cancel");
    cancelButton.addActionListener (this);
    helpButton = new JButton ("Help");
    helpButton.addActionListener (this);
    addButton = new JButton ("Add");
    addButton.addActionListener (this);
    deleteButton = new JButton ("Delete");
    deleteButton.addActionListener (this);
    editButton = new JButton ("Edit");
    editButton.addActionListener (this);

    dialog.setTitle ("ID List");
    promptLabel.setText ("Enter or Edit IDs");

    south.add (okButton);
    south.add (cancelButton);
    south.add (addButton);
    south.add (deleteButton);
    south.add (editButton);
    south.add (helpButton);
}

```

```

    }

    public static void openDialog (Vector v)
    {
        vector = (Vector) v.clone ();
        setListElements ();
        inputArea.requestFocus ();
        dialog.setVisible (true);
    }

    public static void setListElements ()
    {
        model.removeAllElements ();
        for (Enumeration enum = vector.elements (); enum.hasMoreElements ();)
        {
            model.addElement (enum.nextElement ());
        }
    }

    public static Vector getIDList ()
    {
        return (Vector) vector.clone ();
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == okButton)
        {
            vector.removeAllElements ();
            for (Enumeration enum = model.elements (); enum.hasMoreElements ();)
            {
                vector.addElement (enum.nextElement ());
            }
            dialog.setVisible(false);
            parentFrame.setSaveRequired (true);
        }
        else if (e.getSource () == cancelButton)
        {
            dialog.setVisible(false);
        }
        else if (e.getSource () == addButton)
        {
            String newId = showInputDialog ();
            if (newId != null && newId.length () != 0)
            {
                if (GrammarCheck.isValid (newId, GrammarCheck.OP_ID))
                {
                    model.addElement (newId);
                    inputArea.setSelectedValue (newId, true);
                }
                else
                {
                    showErrorDialog ("Illegal id value entered");
                }
            }
        }
        else if (e.getSource () == deleteButton)
        {
            int index = inputArea.getSelectedIndex ();
            if (index >= 0) // If there is a selected elements
                model.removeElementAt (index);
            else
                showErrorDialog ("Please select an Id to delete");
        }
        else if (e.getSource () == editButton)
        {
            int index = inputArea.getSelectedIndex ();
            if (index >= 0)
            {
                String editedId = showEditDialog ((String)
                    inputArea.getSelectedValue ());
                if (editedId != null && editedId.length () != 0)
                {
                    if (GrammarCheck.isValid (editedId, GrammarCheck.OP_ID))
                        model.setElementAt (editedId, index);
                    else
                        showErrorDialog ("Illegal Id value enetered");
                }
                else if (editedId.length () == 0)
            }
        }
    }

```

```

        model.removeElementAt (index);
    }
    else
        showErrorDialog ("Please select an Id to edit");
    }
    else if (e.getSource () == helpButton)
    {
    }
}

public void showErrorDialog (String str)
{
    JOptionPane.showMessageDialog (dialog, str, "Error Message"
                                   , JOptionPane.ERROR_MESSAGE);
}

public String showInputDialog ()
{
    return JOptionPane.showInputDialog (dialog, "Enter new Id"
                                       , "ID Input Dialog", JOptionPane.QUESTION_MESSAGE);
}

public String showEditDialog (String initial)
{
    return (String) JOptionPane.showInputDialog (dialog, "Edit Id"
                                                , "ID Edit Dialog", JOptionPane.QUESTION_MESSAGE, null,
                                                null
                                                , initial);
}
}

package caps.GraphEditor;

import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;

public class Popup extends JPopupMenu
{
    JMenuItem decomposeMenuItem = new JMenuItem ("Decompose");

    JMenuItem fontMenuItem = new JMenuItem ("Font");

    JMenuItem colorMenuItem = new JMenuItem ("Color");

    JMenuItem deleteMenuItem = new JMenuItem ("Delete");

    JMenuItem propMenuItem = new JMenuItem ("Properties");

    DrawPanel panel;

    public Popup (DrawPanel parent)
    {
        super ();

        panel = parent;

        add (propMenuItem);
        addSeparator ();
        add (decomposeMenuItem);
        addSeparator ();
        add (deleteMenuItem);
        addSeparator ();
        add (fontMenuItem);
        add (colorMenuItem);

        decomposeMenuItem.addActionListener (parent);
        fontMenuItem.addActionListener (parent);
        colorMenuItem.addActionListener (parent);
        deleteMenuItem.addActionListener (parent);
        propMenuItem.addActionListener (parent);
    }
    /**
     * 4/19/00 SYT
     * disable decompose
     */
    public void setDecomposeManuItem(boolean b)
    {

```

```

        if(!b)
        {
            decomposeMenuItem.setEnabled(false);
        }
    }
    public JMenuItem getDecomposeMenuItem ()
    {
        return decomposeMenuItem;
    }

    public JMenuItem getFontMenuItem ()
    {
        return fontMenuItem;
    }

    public JMenuItem getColorMenuItem ()
    {
        return colorMenuItem;
    }

    public JMenuItem getDeleteMenuItem ()
    {
        return deleteMenuItem;
    }

    public JMenuItem getPropMenuItem ()
    {
        return propMenuItem;
    }

    public void showPopupMenu (boolean isEdge, int x, int y)
    {
        //edge SYT 12/29/99
        if (isEdge)
        {
            decomposeMenuItem.setEnabled (false);
            colorMenuItem.setEnabled (false);
        }
        //vertex SYT 12/29/99
        else
        {
            decomposeMenuItem.setEnabled (true);
            colorMenuItem.setEnabled (true);
        }
        pack ();
        //Display the popupmenu at the position x,y
        //in the coordinate space of the component invoker.
        // SYT 12/28/99
        show (panel, x, y);
    }

} // End of the class Popup
package caps.GraphEditor;

import java.awt.print.*;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.BasicStroke;
import java.awt.geom.*;
import caps.Psdl.*;
import java.util.Enumeration;
import java.util.Vector;

public class PrintJob implements Runnable, Printable, Pageable
{
    Vector printablePages;

    PrinterJob printJob;

    PageFormat format;

    DrawPanel panel;

    int orientation;

    public PrintJob (DrawPanel p, Vertex root)
    {

```

```

panel = p;

printablePages = new Vector (0, 2);

orientation = PageFormat.PORTRAIT;

DataFlowComponent d;
for (Enumeration enum = root.breadthFirstEnumeration ()
; enum.hasMoreElements ();)
{
    d = (DataFlowComponent) enum.nextElement ();
    if (d instanceof Vertex && !d.isLeaf ())
        printablePages.addElement ((Vertex) d);
}

public void run ()
{
    printJob = PrinterJob.getPrinterJob();
    printJob.setPageable (this);
    PageFormat f = printJob.defaultPage ();
    try
    {
        format = printJob.defaultPage ();
        format = printJob.pageDialog (f);
        if (!f.equals (format))
        {
            //if cancel is not selected
            printJob.print();
        }
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
}

public static void print (DrawPanel p, Vertex root)
{
    PrintJob newJob = new PrintJob (p, root);
    new Thread (newJob).start ();
}

public int print (Graphics g, PageFormat f, int pi)
{
    if (pi >= printablePages.size ())
    {
        return Printable.NO_SUCH_PAGE;
    }
    Graphics2D g2D = (Graphics2D) g;

    double scale = f.getImageableWidth () / (DrawPanel.WIDTH + 20);
    if (orientation == PageFormat.PORTRAIT)
    {
        g2D.translate ((f.getWidth () - DrawPanel.WIDTH * scale) / 2,
            (f.getHeight () - DrawPanel.HEIGHT * scale) /
2);
    }
    else
    {
        scale = f.getImageableWidth () / (double) (DrawPanel.HEIGHT +
35 + 10 + 10);
        g2D.translate (((DrawPanel.WIDTH * scale - f.getWidth ()) / 2
+
f.getImageableWidth () +
            f.getImageableX () - (25 * scale)),
            (f.getImageableY () + (20 * scale)));
        g2D.rotate (Math.toRadians (90));
    }

    g2D.scale (scale, scale);
    // Draws components into graphics device
    panel.setParentVertex ((Vertex) printablePages.elementAt (pi), g2D);

    g2D.setStroke (new BasicStroke (1.5f));
    // bounding rectangle around the prototype
    g2D.draw (new Rectangle2D.Double (-5, -5, 1029, 773));
    g2D.setStroke (new BasicStroke (1f));
}

```



```

        g2D.drawString ("Parent Vertex : " + ((Vertex) printablePages
                        .elementAt (pi)).getLabel (),      0, -30);

        return Printable.PAGE_EXISTS;
    }

    public int getNumberOfPages ()
    {
        return printablePages.size ();
    }

    public PageFormat getPageFormat (int pageIndex)
    {
        if (format.getOrientation () == PageFormat.LANDSCAPE)
        {
            orientation = PageFormat.LANDSCAPE;
            Paper p = new Paper ();
            format.setOrientation (PageFormat.PORTRAIT);
            format.setPaper (p);
        }
        else
        {
            Paper p = new Paper ();           // Disable margin settings
            format.setPaper (p);              // They will have no effect
        }
        return format;
    }

    public Printable getPrintable (int pageIndex)
    {
        return this;
    }

} // End of the class PrintJob
package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;
/**
 * Status bar for saving : Save required or Save not required
 * @ author Shen-Yi
 * @ version 1.1
 */
public class StatusBar extends JLabel implements MouseListener
{
    Editor parent;

    public StatusBar (Editor e)
    {
        super ("Save not required");

        parent = e;
    }

    public void mouseEntered (MouseEvent e)
    {
        setText (((AbstractButton) e.getSource ()).getActionCommand ());
    }

    public void mouseExited (MouseEvent e)
    {
        if (parent.isSaveRequired ())
            setText ("Save required");
        else
            setText ("Save not required");
    }

    public void mouseClicked (MouseEvent e)
    {
    }

    public void mousePressed (MouseEvent e)
    {
    }
}

```

```

        public void mouseDragged (MouseEvent e)
        {
        }

        public void mouseReleased (MouseEvent e)
        {
        }

    } // End of the class StatusBar
package caps.GraphEditor ;
import java.awt.*;
import java.awt.event.*;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.*;
import java.util.*;
import caps.Psdl.*;

/**
 * check stream consistence table: Local or Global
 * @ author Shen-Yi Tao
 * @ version 1.0
 */
public class StreamCheckTable extends JFrame
{
    JPanel jPanel1 = new JPanel();
    JButton jButton1 = new JButton();
    JTable table;
    Vector localVector;
    public StreamCheckTable(Vector local )
    {
        super( ("Inconsistence found in Stream: "
            + ((Edge)(local.firstElement())).getLabel()) + ". Select one raw      and click
OK");

        String[] columnNames = {"Stream Type",
                                "State Stream?",
                                "Initial Value",
                                "Latent Value",
                                "Latent Unit"
                                };

        localVector = local;
        table = new JTable(saveToDate(localVector), columnNames);
        table.setPreferredScrollableViewportSize(new Dimension(500, 70));

        //Create the scroll pane and add the table to it.
        JScrollPane scrollPane = new JScrollPane(table);

        //Add the scroll pane to this window.
        getContentPane().add(scrollPane, BorderLayout.CENTER);
        //add OK button
        jButton1.setText("OK");
        jButton1.addActionListener(new      java.awt.event.ActionListener() {

            public void actionPerformed(ActionEvent e)
            {
                jButton1_actionPerformed(e);
            }
        });
        this.getContentPane().add(jPanel1, BorderLayout.SOUTH);
        jPanel1.add(jButton1, null);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
                dispose();
            }
        });

        pack();
        setVisible(true);
    }

    void jButton1_actionPerformed(ActionEvent e)
    {
        int selected = table.getSelectedRow();
        if(selected != -1)

```

```

{
    //update data
    //0 is the first row
    Edge selectedStream =
    (Edge) (localVector.elementAt(selected));
    String streamType = selectedStream.getStreamType();
    boolean isState = selectedStream.isStateStream();
    String initialVal = selectedStream.getInitialValue();
    PSDTime latency = selectedStream.getMet();
    for(Enumeration en = localVector.elements() ;
    en.hasMoreElements();)
    {
        Edge localEdge = (Edge) (en.nextElement());
        localEdge.setStreamType(streamType);
        localEdge.setStateStream(isState);
        localEdge.setInitialValue(initialVal);
        localEdge.setMet(latency);
    }
    dispose();
}
}

public Object[][] saveToDate(Vector local)
{
    Vector localVector = local;
    int totalRow = localVector.size();
    Object [][] data = new Object[totalRow][5];
    int row = 0 ;
    //set data[][] SYT
    for(Enumeration e = localVector.elements() ; e.hasMoreElements();)
    {
        Edge localEdge = (Edge) (e.nextElement());

        for(int col=0 ; col<5 ; col++)
        {
            switch(col)
            {
                case 0:
                    data[row][0] = localEdge.getStreamType();
                    break;

                case 1:
                    if(localEdge.isStateStream())
                        data[row][1] = "YES";
                    else
                        data[row][1] = "NO";
                    break;

                case 2:
                    data[row][2] = localEdge.getInitialValue();
                    break;
                case 3:
                    if (localEdge.getMet() != null)
                        data[row][3] =
Integer.toString(localEdge.getMet().getTimeValue());
                    else
                        data[row][3] = "";
                    break;
                case 4:
                    if (localEdge.getMet() != null)
                    {
                        if( localEdge.getMet().getTimeUnits() == 0)
                            data[row][4] = "microsec";
                        else if( localEdge.getMet().getTimeUnits() == 1)
                            data[row][4] = "ms";
                        else if( localEdge.getMet().getTimeUnits() == 2)
                            data[row][4] = "sec";
                        else if( localEdge.getMet().getTimeUnits() == 3)
                            data[row][4] = "min";
                        else
                            data[row][4] = "hours";
                    }
                    else
                        data[row][4] = "";
                    break;
            }
        }
    }
}

```

```

        row++;
    }
    return data;
}

public StreamCheckTable()
{
    try
    {
        jbInit();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

private void jbInit() throws Exception
{
    this.setFont(new java.awt.Font("Dialog", 0, 14));
}

} //end of class
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import caps.Parser.GrammarCheck;

public class TextEditor implements ActionListener
{
    private static JDialog dialog;

    private static JPanel south;

    private static final int WIDTH = 400;

    private static final int HEIGHT = 300;

    private static int grammarKind;

    protected static JButton okButton;
    protected static JButton cancelButton;
    protected static JButton helpButton;

    protected static JTextArea inputArea;

    protected static JLabel promptLabel;

    static boolean allowsEmptyString;

    static String text;

    protected Editor parentFrame;

    public TextEditor (Editor parent)
    {
        parentFrame = parent;
        dialog = new JDialog (parentFrame, true);
        dialog.setSize (WIDTH, HEIGHT);
        dialog.setResizable (false);
        dialog.setVisible (false);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        dialog.setLocation ((screenSize.width - dialog.getWidth ()) / 2,
            (screenSize.height - dialog.getHeight ()) / 2);
        initialize ();
    }

    protected void initialize ()
    {
        dialog.getContentPane ().setLayout (new BorderLayout(5, 5));

        south = new JPanel ();
        dialog.getContentPane ().add (south, BorderLayout.SOUTH);
    }
}

```

```

        promptLabel = new JLabel ();
        dialog.getContentPane ().add (promptLabel, BorderLayout.NORTH);

        inputArea = new JTextArea ();
        inputArea.setLineWrap (true);
        inputArea.setBorder (BorderFactory.createLoweredBevelBorder ());
        JScrollPane p = new JScrollPane (inputArea);
        p.setBackground (Color.lightGray);
        dialog.getContentPane ().add (p, BorderLayout.CENTER);

        okButton = new JButton ("OK");
        okButton.addActionListener (this);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener (this);

        helpButton = new JButton ("Help");
        helpButton.addActionListener (this);

        south.add (okButton);
        south.add (cancelButton);
        south.add (helpButton);
    }

    public static void openDialog (String title, String prompt, String      str , int
kind, boolean flag)
    {
        dialog.setTitle (title);
        promptLabel.setText (prompt);
        text = str;
        inputArea.setText (str);
        inputArea.requestFocus ();
        grammarKind = kind;
        allowsEmptyString = flag;
        dialog.setVisible (true);
    }

    public static String getString ()
    {
        text = text.trim ();
        return text;
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == okButton) {
            boolean errorStatus = false;
            String str = inputArea.getText ();
            if (str.length () != 0 || !allowsEmptyString) {
                if (GrammarCheck.isValid (str, grammarKind))
                    text = inputArea.getText ();
                else {
                    showErrorDialog ("Illegal value entered");
                    errorStatus = true;
                }
            }
            else {
                text = "";
            }
            if (!errorStatus) {
                dialog.setVisible(false);
                parentFrame.setSaveRequired (true);
            }
        }
        else if (e.getSource () == cancelButton) {
            dialog.setVisible(false);
        }
        else if (e.getSource () == helpButton) {
        }
    }

    public void showErrorDialog (String str)
    {
        JOptionPane.showMessageDialog (dialog, str, "Error Message"
, JOptionPane.ERROR_MESSAGE);
    }

```

```

    }
    package caps.GraphEditor ;
    import javax.swing.JTable;
    import javax.swing.JScrollPane;
    import javax.swing.JPanel;
    import javax.swing.JFrame;
    import java.awt.*;
    import java.awt.event.*;
    import javax.swing.tree.DefaultMutableTreeNode;
    public class TimingCheckTable extends JFrame
    {

        public TimingCheckTable(Object[][] data ,DefaultMutableTreeNode root)
        {
            super("TIMING CHECK TABLE for "+ root.toString());

            String[] columnNames = {"Vertex Name",
                                    "Timing Type",
                                    "1.MET <= FW",
                                    "2.MET <= PER",
                                    "3.MET <= MRT",
                                    "4.MET <= MCP",
                                    "Pass/Fail"
                                   };

            final JTable table = new JTable(data, columnNames);
            table.setPreferredScrollableViewportSize(new Dimension(500, 70));

            //Create the scroll pane and add the table to it.
            JScrollPane scrollPane = new JScrollPane(table);

            //Add the scroll pane to this window.
            getContentPane().add(scrollPane, BorderLayout.CENTER);

            addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    dispose();
                }
            });

            pack();
            setVisible(true);
        }
    }

    package caps.GraphEditor;

    import java.awt.*;
    import javax.swing.*;
    import java.awt.event.*;
    //add 8/19/00 SYT
    import javax.swing.tree.DefaultMutableTreeNode;
    /**
     * Confirm before timing type are being modified on the sub-level Vertexes
     * @author Shen-Yi Tao
     * @version 1.0
     */
    public class TimingTypeChangedDlg extends JDialog
    {
        JPanel mainPanel = new JPanel();
        BorderLayout borderLayout1 = new BorderLayout();
        JPanel jPanel1 = new JPanel();
        JPanel jPanel2 = new JPanel();
        JLabel label = new JLabel();
        JButton OKJButton = new JButton();
        JButton CancelJButton = new JButton();
        VertexProperties VP;
        BorderLayout borderLayout2 = new BorderLayout();
        JLabel jLabel1 = new JLabel();
        //1: timingCombo change, 2: operatorCombo change
        int dialogType = 1;
        public TimingTypeChangedDlg(VertexProperties temp, int type)
        {
            dialogType = type;
            VP = temp ;
            VP.setVisible(false);
        }
    }

```

```

    try
    {
        jbInit();
        pack();
    }
    catch(Exception ex)
    {
        ex.printStackTrace();
    }
}

void jbInit() throws Exception
{
    mainPanel.setLayout(borderLayout1);
    if(dialogType == 1)
        label.setText("WARNING!! All subcomponents will be change to
        Default setup."
            );
    else
        label.setText("WARNING!! All children will change to Terminator" );

    label.setIcon
        (new
        ImageIcon(caps.Caps.class.getResource("Images/headImage.gif")));

    OKJButton.setFont(new java.awt.Font("Dialog", 3, 14));
    OKJButton.setPreferredSize(new Dimension(81, 33));
    OKJButton.setText("OK");
    OKJButton.addActionListener(new java.awt.event.ActionListener()
    {

        public void actionPerformed(ActionEvent e)
        {
            OKJButton_actionPerformed(e);
        }
    });
    CancelJButton.setFont(new java.awt.Font("Dialog", 3, 14));
    CancelJButton.setPreferredSize(new Dimension(81, 33));
    CancelJButton.setText("Cancel");
    CancelJButton.addActionListener(new java.awt.event.ActionListener()
    {

        public void actionPerformed(ActionEvent e)
        {
            CancelJButton_actionPerformed(e);
        }
    });
    mainPanel.setPreferredSize(new Dimension(420, 120));
    jPanel1.setLayout(borderLayout2);
    if(dialogType == 1)
        jLabel1.setText("                (This timing type and MET = 0)");
    else
        jLabel1.setText("");
    getContentPane().add(mainPanel);
    mainPanel.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(OKJButton, null);
    jPanel2.add(CancelJButton, null);
    mainPanel.add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(label, BorderLayout.CENTER);
    jPanel1.add(jLabel1, BorderLayout.SOUTH);

    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
        frameSize.width = screenSize.width;
    //change 2/9/00 SYT
    setLocation((screenSize.width - frameSize.width) / 4
        , (screenSize.height - frameSize.height) / 3);
    setVisible(true);
}

void OKJButton_actionPerformed(ActionEvent e)
{

```

```

        switch(dialogType)
        {
        case 1:
            if(VP.currentTimingType == VP.getTimingCombo().getSelectedIndex())
                VP.isTimingTypeChanged = false;
            else
            {
                //NON-TIME-CRITICAL
                if(VP.getTimingCombo().getSelectedIndex() == 0 )
                    VP.isTimingTypeChanged = false;
                else
                    VP.isTimingTypeChanged = true;
            }
            VP.currentTimingType = VP.getTimingCombo().getSelectedIndex();
            VP.resetTiming(true);
            VP.setVisible(true);
            dispose();
            break;
        case 2:
            VP.isVertexTypeChanged = true;
            VP.setVisible(true);
            dispose();
            break;
        }
    }

    void CancelJButton_actionPerformed(ActionEvent e)
    {
        switch(dialogType)
        {
        case 1:
            VP.getTimingCombo().setSelectedIndex(VP.currentTimingType);
            VP.setVisible(true);
            dispose();
            break;
        case 2:
            VP.isVertexTypeChanged = false;
            VP.setVisible(true);
            dispose();
            break;
        }
    }
}

package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.Border;
import caps.Psdl.DataTypes;
import caps.Psdl.Vertex;
import caps.Parser.GrammarCheck;
import java.util.Vector;
import caps.Builder.*;
import java.io.StringReader;
import javax.swing.tree.DefaultMutableTreeNode;
import java.util.Enumeration;
//add 7/29/00
import javax.swing.tree.DefaultMutableTreeNode;

/**
 * The main toolbar for the prototyping events.
 * Also handles the events associated with the toolbar buttons.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class ToolBar extends JToolBar implements ActionListener
{
    /**
     * location of the CAPS classes
     */
    private String CAPSJavaHome;

    //add 1/28/00 SYT
    private JButton undo;

```



```

//add 1/28/00 SYT
private JButton redo;
//add 7/29/00 SYT
private JButton timingChecker;

//private JButton decompose;

//reuse this if need a gotoParent button close at 2/28/00 SYT
//private JButton goToParent;

/**
 * Initiates the 'Operator' event
 *
 * move initialization of operator to the ToolBar constructor
 */
//private JButton operator = new JButton
//      (new ImageIcon ("caps/Images/operator.gif"));
private JButton operator;

/**
 * Initiates the 'Terminator' event
 *
 * move initialization of terminator to the ToolBar constructor
 */
//private JButton terminator = new JButton
//      (new ImageIcon ("caps/Images/terminator.gif"));
private JButton terminator;

/**
 * Initiates the 'Stream' event
 *
 * move initialization of Stream to the ToolBar constructor
 */
//private JButton stream = new JButton
//      (new ImageIcon ("caps/Images/streams.gif"));
private JButton stream ;

/**
 * Initiates the 'Select' event
 *
 * move initialization of Select to the ToolBar constructor
 */
//private JButton select = new JButton
//      (new ImageIcon ("caps/Images/select.gif"));
private JButton select;

/**
 * Initiates the 'Types' event
 *
 * move initialization of Types to the ToolBar constructor
 */
//private JButton types = new JButton
//      (new ImageIcon ("caps/Images/types.gif"));
private JButton types;

/**
 * Initiates the 'Parent Specs' event
 *
 * move initialization of parentSpec to the ToolBar constructor
 */
//private JButton parentSpecs = new JButton
//      (new ImageIcon ("caps/Images/parentSpec.gif"));
private JButton parentSpecs;

/**
 * Initiates the 'Timers' event
 *
 * move initialization of Timers to the ToolBar constructor
 */
//private JButton timers = new JButton
//      (new ImageIcon ("caps/Images/timers.gif"));
private JButton timers;

/**
 * Initiates the 'Graph Desc' event
 *
 * move initialization of Graph Desc to the ToolBar constructor
 */
//private JButton graphDesc = new JButton
//      (new ImageIcon ("caps/Images/graphDesc.gif"));
private JButton graphDesc;

```

```

/**
 * the JFrame that is the owner of this toolbar.
 */
protected Editor parentFrame;

/**
 * Constructs a new ToolBar object
 *
 * @param frame The parent frame of this toolbar object.
 */
public ToolBar (Editor frame)
{
    parentFrame = frame;

    // added code to initialize CAPS class location
    CAPSJavaHome = System.getProperty ("CAPSJavaHome");
    if (CAPSJavaHome == null)
    {
        CAPSJavaHome = ".";
    }

    operator = new JButton
        (new ImageIcon (CAPSJavaHome +      "/caps/Images/operator.gif"));
    add (operator);

    terminator = new JButton
        (new ImageIcon (CAPSJavaHome +      "/caps/Images/terminator.gif"));
    add (terminator);

    stream = new JButton
        (new ImageIcon (CAPSJavaHome +      "/caps/Images/streams.gif"));
    add (stream);

    select = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/select.gif"));
    add (select);

    types = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/types.gif"));
    add (types);

    parentSpecs = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/parentSpec.gif"));
    add (parentSpecs);

    timers = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/timers.gif"));
    add (timers);

    graphDesc = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/graphDesc.gif"));
    add (graphDesc);

    //reuse this if need a gotoParent button  close at 2/28/00 SYT
    //goToParent = new JButton
    // (new ImageIcon (CAPSJavaHome + "/caps/Images/goToParent.gif"));

    //add 1/28/00 SYT
    undo = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/undo.gif"));
    add (undo);
    //add 1/28/00 SYT
    redo = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/redo.gif"));
    add (redo);

    timingChecker = new JButton
        (new ImageIcon (CAPSJavaHome + "/caps/Images/timerChecker.gif"));

    add (timingChecker);
    //add 2/15/00 SYT
    //decompose.setToolTipText("decompose");

    //reuse this if need a gotoParent button  close at 2/28/00 SYT
    //goToParent.setToolTipText("go to parent");

```

```

//add 1/28/00 SYT
undo.setToolTipText("Undo");
redo.setToolTipText("Redo");
timingChecker.setToolTipText("Timing Checker");

//add 1/28/00 SYT
//decompose.setEnabled(false);

//reuse this if need a gotoParent button close at 2/28/00 SYT
//goToParent.setEnabled(false);

undo.setEnabled(false);
redo.setEnabled(false);
timingChecker.setEnabled(true);

operator.setToolTipText ("Draw an operator");
terminator.setToolTipText ("Draw a terminator");
stream.setToolTipText ("Draw a stream");
select.setToolTipText ("Select");
types.setToolTipText ("Types");
parentSpecs.setToolTipText ("Parent specifications");
timers.setToolTipText ("Timers");
graphDesc.setToolTipText ("Graph Description");

operator.setFocusPainted (false);

operator.setActionCommand ("Draws an operator into the drawing area");
terminator.setActionCommand ("Draws a terminator into the drawing area");
stream.setActionCommand ("Draws a stream into the drawing area");
select.setActionCommand ("Selects a component from the drawing area");
types.setActionCommand ("Opens the text editor to edit data types");
parentSpecs.setActionCommand ("Opens the text editor to edit parent specifications");
timers.setActionCommand ("Opens the id list editor to edit timers");
graphDesc.setActionCommand ("Opens the text editor to edit the graph description");

operator.addMouseListener (parentFrame.getStatusBar ());
terminator.addMouseListener (parentFrame.getStatusBar ());
stream.addMouseListener (parentFrame.getStatusBar ());
select.addMouseListener (parentFrame.getStatusBar ());
types.addMouseListener (parentFrame.getStatusBar ());
parentSpecs.addMouseListener (parentFrame.getStatusBar ());
timers.addMouseListener (parentFrame.getStatusBar ());
graphDesc.addMouseListener (parentFrame.getStatusBar ());

//add 2/15/00 SYT
//decompose.addActionListener(this);

//reuse this if need a gotoParent button close at 2/28/00 SYT
//goToParent.addActionListener(this);

//add 1/28/00 SYT
undo.addActionListener (this);
//add 1/28/00 SYT
redo.addActionListener (this);
timingChecker.addActionListener(this);

operator.addActionListener (this);
terminator.addActionListener (this);
stream.addActionListener (this);
select.addActionListener (this);
types.addActionListener (this);
parentSpecs.addActionListener (this);
timers.addActionListener (this);
graphDesc.addActionListener (this);
}

//add 2/15/00 SYT
//this will be used for setting button.
public JButton getStreamButton()
{
    return stream;
}

//add 2/15/00 SYT
//this will be used for setting button.
/*public JButton getDecomposeButton()

```

```

{
    return decompose;
}*/

//reuse this if need a gotoParent button close at 2/28/00 SYT
//this will be used for setting button.
/*public JButton getGoToParentButton()
{
    return goToParent;
}*/

//add 1/28/00 SYT
//this will be used for setting button.
public JButton getUndoButton()
{
    return undo;
}

//add 1/28/00 SYT
//this will be used for setting button.
public JButton getRedoButton()
{
    return redo;
}

/**
 * This method is called after another operation is finished
 * associated
 * with another button in the toolbar.
 * For example, When an operator is drawn on the DrawPanel
 * , the toolbar will go into select mode.
 */
public void enableSelectButton ()
{
    select.requestFocus ();
}

/**
 * set operator status
 * @ flag - true: set enable, false:set disable
 */
public void setOperatorButton (boolean flag)
{
    if (flag)
        operator.setEnabled (true);
    else
        operator.setEnabled (false);
}

// add 8/22/00 SYT
/**
 * set terminator status
 * @ flag - true: set enable, false:set disable
 */
public void setTerminatorButton (boolean flag)
{
    if (flag)
        terminator.setEnabled (true);
    else
        terminator.setEnabled (false);
}

/**
 * modified by SYT
 * Handles the action events that occur when one of the buttons
 * in this toolbar is selected
 *
 * @param e The associated ActionEvent
 */
public void actionPerformed (ActionEvent e)
{
    //add 2/23/00 SYT
    //parentFrame.getDrawPanel().clearnDrawStream();

    //add 2/15/00 SYT
    /*if(e.getSource () == decompose)
    {
        parentFrame.getDrawPanel().decompose();
    }

```

```

//reuse this if need a gotoParent button close at 2/28/00 SYT
//goToParent.setEnabled(false);
//add 2/15/00 SYT
decompose.setEnabled(false);
}*/

//reuse this if need a gotoParent button close at 2/28/00 SYT
/* if(e.getSource () == goToParent)
{
    parentFrame.getDrawPanel().gotoParent();
    //add 2/15/00 SYT
    goToParent.setEnabled(false);
}*/

//add 1/28/00 SYT
if(e.getSource () == undo)
{
    parentFrame.getDrawPanel().undoPaint();
    //add 7/25/00 SYT
    parentFrame.setSaveRequired(true);
}
if(e.getSource () == redo)
{
    parentFrame.getDrawPanel().redoPaint();
    //add 7/25/00 SYT
    parentFrame.setSaveRequired(true);
}
//add 7/29/00 SYT
if(e.getSource () == timingChecker)
{
    int totalRow = 0;
    for(Enumeration en
= ((DefaultMutableTreeNode) (parentFrame.getRoot()))
        .breadthFirstEnumeration()
        ; en.hasMoreElements(); )
    {
        DefaultMutableTreeNode DMT= (DefaultMutableTreeNode) en.nextElement();
        if(DMT instanceof Vertex)
        {
            totalRow++;
        }
    }

    Object [][] data = new Object[totalRow][7];

    int row = 0 ;
    //set data[][] SYT
    for(Enumeration en = ((DefaultMutableTreeNode) (parentFrame.getRoot()))
        .breadthFirstEnumeration()
        ; en.hasMoreElements(); )
    {
        DefaultMutableTreeNode DMT= (DefaultMutableTreeNode) en.nextElement();
        // I don't want a displayed Root
        // ,and print a blank row on the end. SYT
        if(DMT instanceof Vertex
            && !(((DefaultMutableTreeNode) DMT).isRoot() ) )
        {
            boolean isFailed = false;

            for(int col=0 ; col<7 ; col++)
            {
                switch(col)
                {
                    case 0:
                        data[row][0] = ((Vertex) DMT).getLabel();
                        break;

                    case 1:
                        if(((Vertex) DMT).getTimingType() == 0 )
                            data[row][1] = "NON_TIME_CRITICAL";
                        else if(((Vertex) DMT).getTimingType() == 1 )
                            data[row][1] = "PERIODIC";
                        else
                            data[row][1] = "SPORADIC";
                }
            }
        }
    }
}

```

```

        break;

    case 2:
        if( ((Vertex)DMT).getTimingType() == 1 )
        {
            if( ((Vertex)DMT).getFinishWithin() == null )
                data[row][2] = "FW is NONE";
            else if( ((Vertex)DMT).getMet().getTimeInSeconds() <=
                ((Vertex)DMT).getFinishWithin
                ().getTimeInSeconds() )
                data[row][2] = "Y";
            else
            {
                data[row][2] = "N";
                isFailed = true;
            }
        }
        else
            data[row][2] = "";
        break;

    case 3:
        if( ((Vertex)DMT).getTimingType() == 1 )
        {
            if( ((Vertex)DMT).getPeriod() == null )
                data[row][3] = "PER is NONE";
            else if( ((Vertex)DMT).getMet().getTimeInSeconds() <=
                ((Vertex)DMT).getPeriod().getTimeInSeconds() )
                data[row][3] = "Y";
            else
            {
                data[row][3] = "N";
                isFailed = true;
            }
        }
        else
            data[row][3] = "";
        break;

    case 4:
        if( ((Vertex)DMT).getTimingType() == 2 )
        {
            if( ((Vertex)DMT).getMrt() == null )
                data[row][4] = "MRT is NONE";
            else if( ((Vertex)DMT).getMet().getTimeInSeconds() <=
                ((Vertex)DMT).getMrt().getTimeInSeconds() )
                data[row][4] = "Y";
            else
            {
                data[row][4] = "N";
                isFailed = true;
            }
        }
        else
            data[row][4] = "";
        break;

    case 5:
        if( ((Vertex)DMT).getTimingType() == 2 )
        {
            if( ((Vertex)DMT).getMcp() == null )
                data[row][5] = "MCP is NONE";
            else if( ((Vertex)DMT).getMet().getTimeInSeconds() <=
                ((Vertex)DMT).getMcp().getTimeInSeconds() )
                data[row][5] = "Y";
            else
            {
                data[row][5] = "N";
                isFailed = true;
            }
        }
        else
            data[row][5] = "";
        break;

```

```

        case 6:
            if(isFailed)
                data[row][6] = "F";
            else
                data[row][6] = "P";
        }
    }
    row++;
}

new TimingCheckTable(data
    , (DefaultMutableTreeNode) (parentFrame.getRoot()) );
}

if (e.getSource () == operator)
{
    operator.setFocusPainted (true);
    parentFrame.getDrawPanel ().setSelectionMode (false);
    // DrawPanel.OPERATOR = integer selected_mode. SYT 12/29/99
    parentFrame.getDrawPanel ().setCurrentComponent
        (DrawPanel.OPERATOR);
}
else if (e.getSource () == terminator)
{
    parentFrame.getDrawPanel ().setSelectionMode (false);
    parentFrame.getDrawPanel ().setCurrentComponent
        (DrawPanel.TERMINATOR);
}
else if (e.getSource () == stream)
{
    parentFrame.getDrawPanel ().setSelectionMode (false);
    parentFrame.getDrawPanel ().setCurrentComponent
        (DrawPanel.STREAM);
}
else if (e.getSource () == select)
{
    parentFrame.getDrawPanel ().setSelectionMode (true);
}
else if (e.getSource () == types)
{
    DataTypes types = parentFrame.getDataTypes ();
    TextEditor.openDialog ("Data Types", "View or Edit Data
Types",
                        types.toString (), GrammarCheck.DATA_TYPE,
                        true);
    types.buildTypes (TextEditor.getString ());
}
else if (e.getSource () == parentSpecs)
{
    Vertex parent = parentFrame.getDrawPanel ().getParentVertex
        ();
    TextEditor.openDialog("Parent Vertex Specification"
        , "View or Edit Parent Specification"
        , parent.getSpecification (false)
        , GrammarCheck.CHECK_PARENT_SPEC,
        false);
    PsdlBuilder.setCurrentOp (parent);
    PsdlBuilder.ReInit (new StringReader (TextEditor.getString
        ()));
    try
    {
        PsdlBuilder.operator_spec ();
    }
    catch (ParseException ex)
    { /* This is already caught in GrammarCheck */
    }
}
else if (e.getSource () == timers)
{
    Vertex parent = parentFrame.getDrawPanel ().getParentVertex
        ();
    IdListEditor.openDialog (parent.getTimerList ());
    parent.setTimerList (IdListEditor.getIDList ());
}
else if (e.getSource () == graphDesc)

```

```

    {
        Vertex parent = parentFrame.getDrawPanel ().getParentVertex
        ();
        TextEditor.openDialog ("Informal Graph Description"
            , "View or Edit Informal Graph
            Description"
            , parent.getGraphDesc ()
            , GrammarCheck.INFORMAL_DESCRIPTION,
            true);
        parent.setGraphDesc (TextEditor.getString ());
    }
} // End of the class ToolBar.

```

```

package caps.GraphEditor;
/**
 * Tragger operation conform dialog.
 * author Shen-Yi Tao
 * version 1.0
 */
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class TraggerDlg extends JDialog
{
    VertexProperties VProperties = null;
    JPanel panel1 = new JPanel();
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    JButton jButton1 = new JButton();
    JLabel DeleteJLabel = new JLabel();
    JPanel jPanel2 = new JPanel();
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();

    public TraggerDlg(VertexProperties VP)
    {
        VProperties = VP;
        try
        {
            jbInit();
            pack();
        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
    }

    void jbInit() throws Exception
    {
        panel1.setLayout(borderLayout1);
        jButton1.setFont(new java.awt.Font("Dialog", 3, 14));
        jButton1.setPreferredSize(new Dimension(81, 33));
        jButton1.setText("OK");
        jButton1.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                jButton1_actionPerformed(e);
            }
        });
        DeleteJLabel.setIcon(new ImageIcon(caps.Caps.class
        .getResource("Images/headImage.gif")));
        jPanel2.setLayout(gridBagLayout1);
        jLabel1.setFont(new java.awt.Font("Dialog", 0, 14));
        jLabel1.setText("You are supposed to set up Tragger Stream List.");
        jLabel2.setFont(new java.awt.Font("Dialog", 0, 14));
        jLabel2.setText("or set Tragger as \"Unprotected\"");
        panel1.setPreferredSize(new Dimension(420, 120));
        getContentPane().add(panel1);
    }
}

```



```

panel1.add(jPanel1, BorderLayout.SOUTH);
jPanel1.add(jButton1, null);
panel1.add(deleteJLabel, BorderLayout.WEST);
panel1.add(jPanel2, BorderLayout.CENTER);
jPanel2.add(jLabel2, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
    , GridBagConstraints.CENTER, GridBagConstraints.NONE
    , new Insets(0, 0, 0, 0), 30, 0));
jPanel2.add(jLabel1, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
    , GridBagConstraints.EAST, GridBagConstraints.NONE
    , new Insets(0, 11, 0, 0), 43, 0));
//Center the window. SYT
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = this.getSize();
if (frameSize.height > screenSize.height)
    frameSize.height = screenSize.height;
if (frameSize.width > screenSize.width)
    frameSize.width = screenSize.width;
//change 4/22/00 SYT
setLocation((screenSize.width - frameSize.width) / 4
    , (screenSize.height - frameSize.height) / 3);
setVisible(true);
}

void jButton1_actionPerformed(ActionEvent e)
{
    VProperties.setVisible(true);
    dispose();
}

}
package caps.GraphEditor;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;
import caps.Psdl.*;
//add SYT
import java.util.*;
import java.lang.reflect.*;
/**
 * The treepanel is the place where the hierarchic structure of
 * the prototype is displayed.
 * Have a functionality for duplicating itself.
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class TreePanel extends JTree implements TreeSelectionListener
    , TreeModelListener, MouseListener
{
    //add SYT
    Vertex treeRoot;
    //add 2/8/00
    Vertex parentNode;
    //add 2/8/00
    //Vector childrenVector;
    /**
     * the JFrame that is the owner of this panel.
     */
    Editor parentFrame;

    DefaultTreeModel model;

    /**
     * Constructs a new TreePanel object
     *
     * @param frame The parent frame of this treepanel object.
     */
    public TreePanel (Editor frame, Vertex root)
    {
        super ();

        //add SYT
        treeRoot = root;
        parentFrame = frame;
    }

```

```

//add 2/20/00 SYT
setEditable(true);
//add
addMouseListener(this);
this.getCellEditor().addCellEditorListener(new
CellEditorListener()
{
    public void editingCanceled(ChangeEvent e)
    {

    }
    public void editingStopped(ChangeEvent e)
    {
    }
});

DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer
();
setCellRenderer (renderer);
addTreeSelectionListener (this);

//change. SYT
//model = new DefaultTreeModel (root);
model = new DefaultTreeModel (treeRoot);
model.setAsksAllowsChildren (true);
model.addTreeModelListener (this);

setModel (model);
setCellRenderer (new TreePanelRenderer ());
//allow only a single selection. SYT
getSelectionModel().setSelectionMode
(TreeSelectionMode.SINGLE_TREE_SELECTION);
setShowsRootHandles(true);
setEditable (false);

setAlignmentX (LEFT_ALIGNMENT);
setAlignmentY (TOP_ALIGNMENT);
setBorder (BorderFactory.createEtchedBorder ());

}

//add SYT
public DefaultMutableTreeNode getParentNode()
{
    return (DefaultMutableTreeNode) parentNode ;
}

//add. SYT
public DefaultTreeModel getTreeModel()
{
    return model;
}

// add. SYT
/**
 * clone a tree model which has the same children structure as
 * current
 * treeModel
 */
public DefaultTreeModel cloneTreeModel()
{
    DefaultMutableTreeNode cloneRoot = new DefaultMutableTreeNode() ;
    DefaultTreeModel cloneTreeModel= new DefaultTreeModel(cloneRoot);
    cloneTreeModel.setAsksAllowsChildren(true);

    for(Enumeration e =( (DefaultMutableTreeNode) (model.getRoot()))
        .breadthFirstEnumeration();e.hasMoreElements();)
    {
        DefaultMutableTreeNode DMT =(DefaultMutableTreeNode) e.nextElement();

        if( DMT.getChildCount() != 0 )
        {
            if( DMT.isRoot() )
            {
                cloneOneLevel(model,cloneTreeModel,DMT,true);
            }
        }
    }
}

```

```

        else
        {
            cloneOneLevel(model, cloneTreeModel, DMT, false);
        }
    }
    //mark parent vertex
    noteParentVertex(cloneTreeModel);

    return cloneTreeModel;
}

/**
 * add 3/10/00
 * note current parent vertex
 */
public void noteParentVertex(DefaultTreeModel DTM)
{
    for(Enumeration e = ( (DefaultMutableTreeNode) ( DTM.getRoot() ) )
        .breadthFirstEnumeration(); e.hasMoreElements(); )
    {
        DefaultMutableTreeNode DMTN
            = ( (DefaultMutableTreeNode) (e.nextElement()) );

        if( (DMTN instanceof Vertex) && !(DMTN instanceof External) )
        {
            if( ((Vertex)DMTN).getCloneVertexID()
                == parentFrame.getDrawPanel()
                    .getParentVertex().getCloneVertexID() )
            {
                ((Vertex)DMTN).setIsParent(true);
            }
            else
            {
                ((Vertex)DMTN).setIsParent(false);
            }
        }
    }
}

// add 2/9/00 SYT
/**
 * creat one level tree structure at a time
 * @param thisModel is current tree model
 * @param cModel is the target clone tree model
 * @param parentNode is the parent node of the current level
 * @param isRoot: true for root
 */
public void cloneOneLevel(DefaultTreeModel thisModel
                        ,DefaultTreeModel cModel
                        ,DefaultMutableTreeNode
                        parentNode
                        ,boolean isRoot )
{
    DefaultTreeModel currentModel = thisModel;
    DefaultTreeModel cloneModel = cModel;
    DefaultMutableTreeNode parent = parentNode;
    DefaultMutableTreeNode cloneParent = null;
    boolean isRootFlag = false ;

    //if this is root
    if(isRoot)
        isRootFlag = true;
    else
        isRootFlag = false;

    Vector InEdgesCollection = new Vector(1);
    Vector OutEdgesCollection = new Vector(1);

    int childCount = ( (DefaultMutableTreeNode) parent ).getChildCount();
    //duplicate a tree structure as current tree. SYT

    if(isRootFlag)
    {
        cloneParent = (DefaultMutableTreeNode) ( cloneModel.getRoot() );

        for(int i=0; i<childCount ; i++)

```

```

        {
            DefaultMutableTreeNode childNode =
            (DefaultMutableTreeNode)
            (( (DefaultMutableTreeNode) ( ( (DefaultMutableTreeNode)
parent
            .getChildAt(i) ) ).clone() ) ;

            cloneModel.insertNodeInto(childNode, cloneParent ,i);
        }
    }
else
    {
        //find the clone parent SYT
        for(Enumeration n = ((DefaultMutableTreeNode) (cloneModel.getRoot()))
            .breadthFirstEnumeration(); n.hasMoreElements();)
        {
            DefaultMutableTreeNode DMT = ( DefaultMutableTreeNode )
            ( n.nextElement());
            if( DMT instanceof Vertex )
            {
                if( ((Vertex)DMT).getCloneVertexID() == ((Vertex)parent)
                    .getCloneVertexID() )
                {
                    cloneParent = DMT;
                    for(int i=0; i<childCount ; i++)
                    {
                        DefaultMutableTreeNode childNode =
                        (DefaultMutableTreeNode)(( (DefaultMutableTreeNode)
parent )
                        ( ( (DefaultMutableTreeNode)
                        .getChildAt(i) ) ).clone() ) ;

                        cloneModel.insertNodeInto(childNode, cloneParent ,i);
                    }
                }
            }
        }
        //set edges source and destination (vertex). 1/21/00 SYT

        reconstruct(cloneParent);
    } //end cloneTreeModel

// add SYT
/**
 * prevent memory sharing problem
 * write correct copy of the inner inEdges , outEdges vector inside
 * the Vetex.
 */
public void reconstruct(DefaultMutableTreeNode d)
{
    for(Enumeration e = d.children(); e.hasMoreElements();)
    {
        DataFlowComponent DFC = (DataFlowComponent)
        (e.nextElement());
        //if( ( (DataFlowComponent) (d.getRoot() ) ).getChildAt(i)
//instanceof
        // Vertex)
        if(DFC instanceof Edge)
        {
            //clone points vector. SYT
            ((Edge) DFC).clonePoints() ;

            if (((Edge) DFC).getSource () instanceof External)
            {
                createCloneExternal((Edge)DFC,true);
            }
            if (((Edge) DFC).getDestination () instanceof External)
            {
                createCloneExternal((Edge)DFC,false) ;
            }
        }
    }

    if(DFC instanceof Vertex)
    {

```

```

//change address (slove clone share address problem). SYT
//create theses new empty vertexes
((Vertex)DFC).cloneInEdges();
((Vertex)DFC).cloneOutEdges();
((Vertex)DFC).cloneMetReqmts();
((Vertex)DFC).clonePeriodReqmts();

for(Enumeration x = d.children(); x.hasMoreElements();)
{
    DataFlowComponent EDFC =
    (DataFlowComponent)(x.nextElement() );

    if(EDFC instanceof Edge)
    {
        //modify address(for clone). SYT
        //set source.
        if( ( (Edge)EDFC ).getSource().getCloneVertexID()
            == ((Vertex)DFC).getCloneVertexID() )
        {
            ( (Edge)EDFC ).setSource( (Vertex)DFC );
            //change address SYT
            //updat out edge vector.
            //change address 1/21/00 SYT
            ((Vertex)DFC).addOutEdge((Edge)EDFC);
        }
        if(( (Edge)EDFC
            ).getDestination().getCloneVertexID()
            ==((Vertex)DFC).getCloneVertexID() )
        {
            ( (Edge)EDFC ).setDestination( (Vertex)DFC );
            //for changing address 1/21/00 SYT
            ((Vertex)DFC).addInEdge((Edge)EDFC);
        }

        //reconstructInnerEdge( (Vertex)DFC ,(Edge)EDFC );
    }
}

}
} //end reconstruct

// add 2/12/00 SYT
/**
 * create a clone external for the clone Edge
 * @param ed : an clone Edge
 * @param flag : true for InEdge , false for OutEdge
 */

public void createCloneExternal(Edge ed , boolean flag)
{
    if(flag)
    {
        External cloneExternal =(External)( ed.getSource().clone() );
        cloneExternal.cloneOutEdges();

        //for changing address SYT
        ed.setSource(cloneExternal );
        //updat in edge vector.
        cloneExternal.addOutEdge(ed);
    }
    else
    {
        External cloneExternal =(External)(
            ed.getDestination().clone());

        cloneExternal.cloneInEdges();

        //for changing address SYT
        ed.setDestination(cloneExternal );
        //updat in edge vector.
        cloneExternal.addInEdge(ed);
    }
}

//add 1/16/00 SYT
// update current model as d
public void updateTreeModel(DefaultTreeModel d)

```

```

{
    //clearn child node. SYT
    ( ( DefaultMutableTreeNode) treeRoot ).removeAllChildren();

    //holding temp vector. SYT
    Vector tempV = new Vector();

    for(Enumeration e =
        ( (DefaultMutableTreeNode)(d.getRoot()) ).children()
        ; e.hasMoreElements(); )
    {
        DefaultMutableTreeNode childNode =
            (DefaultMutableTreeNode)(
                e.nextElement());

        tempV.addElement(childNode);
    }

    while(! tempV.isEmpty())
    {
        DefaultMutableTreeNode childNode =
            (DefaultMutableTreeNode)( tempV.firstElement());
        //add and remove 1/22/00 SYT
        ( (DefaultMutableTreeNode) treeRoot ).add(childNode);

        tempV.removeElementAt(0);
    }
    model.reload();
}

//notify that the nodes were inserted. SYT
public void addNewDFC (DataFlowComponent dfc, DataFlowComponent
parent)
{
    int [] index = {parent.getIndex (dfc)};
    model.nodesWereInserted (parent, index);
}

//changed 1/9/00 SYT
/**
 * remove data flow components from screen
 */
public void removeDfc ()
{
    model.reload ();
}

//added 1/8/00 SYT
/**
 * remove all data flow components from screen
 */
public void removeAllDfc()
{
    DefaultMutableTreeNode rootNode =
        ((DefaultMutableTreeNode)model.getRoot());
    rootNode.removeAllChildren();
    model.reload();
}

// added by SYT 3/7/00 ,4/19/00
/**
 * decompose a tree node into it's sub-level
 * @param selectedDfc is the selected DataFlowComponent
 */
public void decompose (DataFlowComponent selectedDfc )
{
    if (selectedDfc==null)
        return;

    DataFlowComponent dfc = selectedDfc;

    if (dfc instanceof Vertex && ((Vertex) dfc).isTerminator ())
        parentFrame.getToolBar ().setOperatorButton (false);
    if (dfc instanceof Vertex && !(dfc instanceof External ) )
    {
        if (dfc.getLabel().indexOf(".") == -1)
            parentFrame.getDrawPanel().setParentVertex((Vertex)dfc, null);
    }
}

```

```

    }

}

// Called whenever the node was selected. 1/9/00 SYT
/**
 * triggered when a tree node had been selected.
 */
public void valueChanged (TreeSelectionEvent e)
{
    TreePath path = e.getPath ();
    DataFlowComponent dfc = (DataFlowComponent)
    path.getLastPathComponent ();
    if (dfc.isRoot ())
        parentFrame.getDrawPanel ().gotoRoot();
    else if (dfc instanceof Vertex && !(dfc instanceof External))
    {
        //8/22/00 SYT
        //critical status: soft
        /* if (!((Vertex)dfc).isTerminator()
           & ((Vertex) dfc).getCriticalStatus() ==2 )
           parentFrame.getToolBar ().setTerminatorButton(false);
        else
           parentFrame.getToolBar ().setTerminatorButton(true);
        */
        if (((Vertex) dfc).isLeaf ()) // If this is an atomic vertex
        {
            parentFrame.getDrawPanel ().changeLevel ((Vertex)
            dfc.getParent ());

            parentFrame.getDrawPanel ().setSelectedDFC (dfc);
            //change 3/7/00 SYT
            //parentFrame.getDrawPanel ().decompose();
            decompose(dfc);

            //reuse this if need a gotoParent button close at 2/28/00
            // SYT

        }
        else
        {
            // If this is composite
            parentFrame.getDrawPanel ().changeLevel ((Vertex) dfc);
        }
        //reuse this if need a gotoParent button close at 2/28/00 SYT
        //parentFrame.getToolBar().getGoToParentButton().setEnabled(false);
    }
    else if (dfc instanceof Edge)
    {
        // If this is an Edge
        parentFrame.getDrawPanel ().changeLevel ((Vertex) dfc.getParent ());
        parentFrame.getDrawPanel ().setSelectedDFC (dfc);

        //reuse this if need a gotoParent button close at 2/28/00 SYT
        // parentFrame.getToolBar().getGoToParentButton().setEnabled(false);
    }
    parentFrame.getDrawPanel ().setMenuBarItems ();
}

public void treeNodesChanged (TreeModelEvent e)
{
}

public void treeNodesInserted (TreeModelEvent e)
{
}

public void treeNodesRemoved (TreeModelEvent e)
{
}

public void treeStructureChanged (TreeModelEvent e)
{
}

public void mousePressed (MouseEvent e)
{
}

```

```

    public void mouseEntered (MouseEvent e)
    {
    }
    public void mouseExited (MouseEvent e)
    {
    }
    public void mouseClicked(MouseEvent e)
    {
    }
    public void mouseReleased (MouseEvent e)
    {
    }
    public void mouseDragged (MouseEvent e)
    {
    }
    public void mouseMoved (MouseEvent e)
    {
    }

} // End of the class TreePanel.

package caps.GraphEditor;

/**
 * The treepanel is the place where the hierarchic structure of
 * the prototype is displayed.
 *
 * @author Shen-Yi Tao
 * @version 1.1
 */

/* Changes:
 *   added CAPSJavaHome private attributes
 *   added code to TreePanelRenderer to initialize CAPSJavaHome
 */

import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JTree;
import javax.swing.tree.TreeCellRenderer;
import javax.swing.tree.DefaultMutableTreeNode;
import java.awt.Component;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import caps.Psdl.*;

public class TreePanelRenderer extends JLabel implements TreeCellRenderer
{

    static protected Font defaultFont;

    static protected ImageIcon termCompositeIcon;

    static protected ImageIcon termAtomicIcon;

    static protected ImageIcon opCompositeIcon;

    static protected ImageIcon opAtomicIcon;

    static protected ImageIcon streamIcon;

    static protected ImageIcon stateStreamIcon;

    /**
     * added CAPSJavaHome to remember location of the CAPS classes
     */

    static protected String CAPSJavaHome;

    /**
     * Color to use for the background when selected.
     */
    static protected final Color SelectedBackgroundColor = Color.lightGray;

```



```

static
{
    try
    {
        defaultFont = new Font("SansSerif", 0, 12);
    }
    catch (Exception e)
    {
    }
    try
    {
        // add code to initialize CAPSJavaHome
        CAPSJavaHome = System.getProperty ("CAPSJavaHome");
        if (CAPSJavaHome == null)
        {
            CAPSJavaHome = ".";
        }

        // added CAPSJavaHome to the following 6 statements
        termCompositeIcon = new ImageIcon (CAPSJavaHome
            + "/caps/Images/termComposite.gif");
        termAtomicIcon = new ImageIcon (CAPSJavaHome
            + "/caps/Images/termAtomic.gif");
        opCompositeIcon = new ImageIcon (CAPSJavaHome
            + "/caps/Images/opComposite.gif");
        opAtomicIcon = new ImageIcon (CAPSJavaHome
            + "/caps/Images/opAtomic.gif");
        streamIcon = new ImageIcon (CAPSJavaHome
            + "/caps/Images/streamIcon.gif");
        stateStreamIcon = new ImageIcon (CAPSJavaHome
            + "/caps/Images/stateStreamIcon.gif");
    }
    catch (Exception e)
    {
        System.out.println("Couldn't load images: " + e);
    }
}

/**
 * Whether or not the item that was last configured is selected.
 */
protected boolean selected;

/**
 * This is messaged from JTree whenever it needs to get the size
 * of the component or it wants to draw it.
 * This attempts to set the font based on value, which will be
 * a TreeNode.
 */
public Component getTreeCellRendererComponent(JTree tree, Object
    value,
    boolean selected, boolean expanded,
    boolean leaf, int row,
    boolean hasFocus)
{
    Font font;
    String stringValue = tree.convertValueToText(value, selected,
        expanded, leaf, row, hasFocus);

    // Set the color and the font based on the SampleData userObject.
    DataFlowComponent userObject = (DataFlowComponent) value;

    // Set the text.
    setText(stringValue);
    // Tooltips used by the tree.
    setToolTipText(stringValue);

    // Set the image.
    if (userObject instanceof Vertex && ((Vertex) userObject).isLeaf
        ())
    {
        if (((Vertex) userObject).isTerminator ())
            setIcon (termAtomicIcon);
        else
            setIcon (opAtomicIcon);
    }
}

```

```

        else if (userObject instanceof Vertex && !((Vertex)
            userObject).isLeaf ())
        {
            if (((Vertex) userObject).isTerminator ())
                setIcon (termCompositeIcon);
            else
                setIcon (opCompositeIcon);
        }
        else if (userObject instanceof Edge && ((Edge)
            userObject).isStateStream ())
            setIcon (stateStreamIcon);
        else
            setIcon (streamIcon);
    /*
        if (hasFocus)
            setForeground(Color.cyan);
        else
            setForeground(userObject.getColor());
        if (userObject.getFont() == null)
            setFont(defaultFont);
        else
            setFont(userObject.getFont());
    */

    /* Update the selected flag for the next paint. */
    this.selected = selected;

    return this;
}

/**
 * paint is subclassed to draw the background correctly. JLabel
 * currently does not allow backgrounds other than white, and it
 * will also fill behind the icon. Something that isn't desirable.
 */
public void paint(Graphics g)
{
    Color bColor;
    Icon currentI = getIcon();

    if (selected)
        bColor = SelectedBackgroundColor;
    else if (getParent() != null)
        // Pick background color up from parent (which will come from
        // the JTree we're contained in).
        bColor = getParent().getBackground();
    else
        bColor = getBackground();
    g.setColor(bColor);
    if (currentI != null && getText() != null)
    {
        int offset = (currentI.getIconWidth() + getIconTextGap());
        g.fillRect(offset, 0, getWidth() - 1 - offset, getHeight() - 1);
    }
    else
        g.fillRect(0, 0, getWidth()-1, getHeight()-1);
    super.paint(g);
}
}

package caps.GraphEditor;

/**
 * Tragger operation conform dialog.
 * author Shen-Yi Tao
 * version 1.0
 */
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class TriggerDlg extends JDialog
{
    VertexProperties VProperties = null;
    JPanel panell = new JPanel();
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    JButton jButton1 = new JButton();

```

```

JLabel DeleteJLabel = new JLabel();
JPanel jPanel2 = new JPanel();
GridBagLayout gridBagLayout1 = new GridBagLayout();
JLabel jLabel1 = new JLabel();
JLabel jLabel2 = new JLabel();

public TriggerDlg(VertexProperties VP)
{
    VProperties = VP;
    try
    {
        jbInit();
        pack();
    }
    catch(Exception ex)
    {
        ex.printStackTrace();
    }
}

void jbInit() throws Exception
{
    panell1.setLayout(borderLayout1);
    jButton1.setFont(new java.awt.Font("Dialog", 3, 14));
    jButton1.setPreferredSize(new Dimension(81, 33));
    jButton1.setText("OK");
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            jButton1_actionPerformed(e);
        }
    });
    DeleteJLabel.setIcon(new ImageIcon(caps.Caps.class
        .getResource("Images/headImage.gif")));
    jPanel2.setLayout(gridBagLayout1);
    jLabel1.setFont(new java.awt.Font("Dialog", 0, 14));
    jLabel1.setText("You are supposed to set up Trigger Stream List.");
    jLabel2.setFont(new java.awt.Font("Dialog", 0, 14));
    jLabel2.setText("or set Trigger as \"Unprotected\"");
    panell1.setPreferredSize(new Dimension(420, 120));
    getContentPane().add(panell1);
    panell1.add(jPanel1, BorderLayout.SOUTH);
    jPanel1.add(jButton1, null);
    panell1.add>DeleteJLabel, BorderLayout.WEST);
    panell1.add(jPanel2, BorderLayout.CENTER);
    jPanel2.add(jLabel2, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
        , GridBagConstraints.CENTER, GridBagConstraints.NONE
        , new Insets(0, 0, 0, 0), 30, 0));
    jPanel2.add(jLabel1, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
        , GridBagConstraints.EAST, GridBagConstraints.NONE
        , new Insets(0, 11, 0, 0), 43, 0));
    //Center the window. SYT
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
        frameSize.width = screenSize.width;
    //change 4/22/00 SYT
    setLocation((screenSize.width - frameSize.width) / 4
        , (screenSize.height - frameSize.height) / 3);
    setVisible(true);
}

void jButton1_actionPerformed(ActionEvent e)
{
    VProperties.setVisible(true);
    dispose();
}

}
/**
 * The UI to set up Vertex property
 * @ author Shen-Yi Tao

```

```

    * @ version 1.1
    */
package caps.GraphEditor;

import javax.swing.*;

import java.awt.event.*;
import java.awt.*;
import caps.Psdl.*;
import caps.Display.DisplayVertex;
import caps.Parser.GrammarCheck;
import java.util.Vector;
import java.util.Enumuration;
//add 7/19/00 SYT
import javax.swing.tree.DefaultMutableTreeNode;

public class VertexProperties extends JDialog implements ActionListener
{
    //add 8/19/00 SYT
    public static boolean isVertexTypeChanged = false;
    //add 7/25/00 SYT
    public static boolean isTimingTypeChanged = false;
    //add 7/23/00 SYT
    public static int currentTimingType = 0;

    public static final int TO_OPERATOR = 0;

    public static final int TO_TERMINATOR = 1;

    public static final int UNCHANGED = 3;

    private int changeStatus;

    Vertex targetVertex;

    DisplayVertex dVertex;

    JPanel namePanel;
    JPanel triggerPanel;
    JPanel timingPanel;
    JPanel guardsPanel;
    JPanel keywordsPanel;
    JPanel okPanel;

    JTextField nameField;
    TextArea ifCondField;
    JTextField metField;
    JTextField periodField;
    JTextField fwField;
    //add 3/31/00 SYT;
    JTextField networkMappingField;
    //add 7/3/00 SYT
    JLabel criticalnessLabel;
    ButtonGroup criticalnessBG;
    JRadioButton hardRB;
    JRadioButton softRB;

    JLabel metLabel;
    JLabel periodLabel;
    JLabel finishWithinLabel;

    JComboBox operatorCombo;
    JComboBox languageCombo;
    JComboBox triggerCombo;
    JComboBox timingCombo;
    JComboBox metUnitsCombo;
    JComboBox periodUnitsCombo;
    JComboBox fwUnitsCombo;

    JButton ifConditionButton;
    JButton triggerReqByButton;
    JButton metReqByButton;
    JButton periodReqByButton;
    JButton fwReqByButton;
    JButton outputGuardsButton;

```

```

JButton exceptionGuardsButton;
JButton exceptionListButton;
JButton timerOpsButton;
JButton keywordsButton;
JButton informalDescButton;
JButton formalDescButton;
JButton okButton;
JButton cancelButton;
JButton helpButton;
JButton triggerStreamsButton;

Editor parentFrame;

Vertex tempVertex;

public VertexProperties (Editor parent)
{
    super (parent, "Vertex Properties", true);

    parentFrame = parent;
    setResizable (false);
    initialize ();
    pack ();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation ((screenSize.width - getWidth ()) / 2,
                (screenSize.height - getHeight ()) / 2);
    // 4-13-99 Eagle change for size testing
    setSize(480, 560);
}

public void initialize ()
{
    Box box = Box.createVerticalBox ();

    GridBagConstraints gbc = new GridBagConstraints ();
    gbc.fill = GridBagConstraints.BOTH;
    gbc.insets = new Insets (1, 2, 1, 2);

    namePanel = new JPanel (new GridBagLayout ());
    namePanel.setBorder (BorderFactory.createTitledBorder (""));
    nameField = new JTextField ();
    //add 3/31/00 SYT
    //change 7/16/00 SYT
    //networkMappingField = new JTextField();
    networkMappingField = new JTextField("local_network");

    operatorCombo = new JComboBox ();
    operatorCombo.addItem ("Operator");
    operatorCombo.addItem ("Terminator");
    operatorCombo.addActionListener (this);
    languageCombo = new JComboBox ();
    languageCombo.addItem ("Ada");
    languageCombo.addItem ("TAE");

    gbc.gridwidth = 1; gbc.gridheight = 1; gbc.gridx = 0; gbc.gridy = 0;
    namePanel.add (new JLabel ("Name :"), gbc);
    gbc.gridwidth = 2; gbc.gridx++;
    namePanel.add (nameField, gbc);
    gbc.gridwidth = 1; gbc.gridx = 3;
    namePanel.add (operatorCombo, gbc);

    gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy++;
    namePanel.add (new JLabel ("Implementation Language :"), gbc);
    gbc.gridwidth = 1; gbc.gridx = 2;
    namePanel.add (languageCombo, gbc);

    //add 3/31/00 SYT
    gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy=2;
    namePanel.add (new JLabel ("Network Mapping :"), gbc);
    gbc.gridwidth = 2; gbc.gridx = 1;
    namePanel.add (networkMappingField, gbc);

    triggerPanel = new JPanel (new GridBagLayout ());
    triggerPanel.setBorder (BorderFactory.createTitledBorder ("Trigger : "));
    triggerCombo = new JComboBox ();
    triggerCombo.addItem ("Unprotected");
    triggerCombo.addItem ("By Some");

```

```

triggerCombo.addItem ("By All");
triggerCombo.addActionListener (this);
triggerStreamsButton = new JButton (" Stream List ");
triggerStreamsButton.addActionListener (this);
ifConditionButton = new JButton ("If Condition");
ifConditionButton.addActionListener (this);
ifCondField = new TextArea ("", 1, 20, TextArea.SCROLLBARS_VERTICAL_ONLY);
ifCondField.setEditable (false);
triggerReqByButton = new JButton (" Required By ");
triggerReqByButton.addActionListener (this);
//gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 0;
//triggerPanel.add (new JLabel ("Trigger :"), gbc);
gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
triggerPanel.add (triggerCombo, gbc);
gbc.gridx = 3;
triggerPanel.add (triggerStreamsButton, gbc);
gbc.gridx = 1; gbc.gridy++;
triggerPanel.add (ifConditionButton, gbc);
gbc.gridwidth = 2; gbc.gridx++;
triggerPanel.add (ifCondField, gbc);
//gbc.gridwidth = 1; gbc.gridx = 4;
//triggerPanel.add (Box.createRigidArea (new Dimension (10, 5)));
gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 2;
triggerPanel.add (triggerReqByButton, gbc);

timingPanel = new JPanel (new GridBagLayout ());
timingPanel.setBorder (BorderFactory.createTitledBorder ("Timing : "));

//add 7/3/00 SYT
criticalnessLabel = new JLabel(" Criticalness :");
criticalnessBG = new ButtonGroup();
hardRB = new JRadioButton("hard",true);
hardRB.addActionListener(this);
softRB = new JRadioButton("soft",false);
softRB.addActionListener(this);

criticalnessBG.add(hardRB);
criticalnessBG.add(softRB);

timingCombo = new JComboBox ();
timingCombo.addItem ("Non-time critical");
timingCombo.addItem ("Periodic");
timingCombo.addItem ("Sporadic");

//timingCombo.addActionListener (this);
//7/25/00 SYT

timingCombo.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        timingCombo_actionPerformed(e);
    }
});

metField = new JTextField ();
metUnitsCombo = getUnitsCombo ();
metReqByButton = new JButton (" Required By ");
metReqByButton.addActionListener (this);
periodField = new JTextField ();

periodUnitsCombo = getUnitsCombo ();
periodReqByButton = new JButton (" Required By ");
periodReqByButton.addActionListener (this);
fwField = new JTextField ();
fwUnitsCombo = getUnitsCombo ();
fwReqByButton = new JButton (" Required By ");
fwReqByButton.addActionListener (this);
metLabel = new JLabel ("MET : ");
periodLabel = new JLabel ("Period : ");
finishWithinLabel = new JLabel ("FinishWithin : ");

gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
timingPanel.add (timingCombo, gbc);
//7/3/00 SYT
gbc.gridwidth = 1; gbc.gridx = 2; gbc.gridy = 0;

```

```

timingPanel.add (criticalnessLabel, gbc);
gbc.gridwidth = 1; gbc.gridx = 3; gbc.gridy = 0;
timingPanel.add (hardRB, gbc);
gbc.gridwidth = 1; gbc.gridx = 4; gbc.gridy = 0;
timingPanel.add (softRB, gbc);

gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 1;
timingPanel.add (metLabel, gbc);
gbc.gridwidth = 1; gbc.gridx = 1;
timingPanel.add (metField, gbc);
gbc.gridwidth = 1; gbc.gridx = 2;
timingPanel.add (metUnitsCombo, gbc);
gbc.gridwidth = 2; gbc.gridx = 3;
timingPanel.add (metReqByButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 2;
timingPanel.add (periodLabel, gbc);
gbc.gridwidth = 1; gbc.gridx = 1;
timingPanel.add (periodField, gbc);
gbc.gridwidth = 1; gbc.gridx = 2;
timingPanel.add (periodUnitsCombo, gbc);
gbc.gridwidth = 2; gbc.gridx = 3;
timingPanel.add (periodReqByButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 3;
timingPanel.add (finishWithinLabel, gbc);
gbc.gridwidth = 1; gbc.gridx = 1;
timingPanel.add (fwField, gbc);
gbc.gridwidth = 1; gbc.gridx = 2;
timingPanel.add (fwUnitsCombo, gbc);
gbc.gridwidth = 2; gbc.gridx = 3;
timingPanel.add (fwReqByButton, gbc);

gbc.insets = new Insets (1, 15, 1, 15);
guardsPanel = new JPanel (new GridBagLayout ());
guardsPanel.setBorder (BorderFactory.createTitledBorder (""));
outputGuardsButton = new JButton (" Output Guards ");
outputGuardsButton.addActionListener (this);
exceptionGuardsButton = new JButton (" Exception Guards ");
exceptionGuardsButton.addActionListener (this);
exceptionListButton = new JButton (" Exception List ");
exceptionListButton.addActionListener (this);
timerOpsButton = new JButton (" Timer Ops ");
timerOpsButton.addActionListener (this);
gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 0;
guardsPanel.add (outputGuardsButton, gbc);
gbc.gridx = 2;
guardsPanel.add (exceptionGuardsButton, gbc);
gbc.gridx = 0; gbc.gridy = 1;
guardsPanel.add (exceptionListButton, gbc);
gbc.gridx = 2;
guardsPanel.add (timerOpsButton, gbc);

gbc.insets = new Insets (1, 2, 1, 2);
keywordsPanel = new JPanel (new FlowLayout ());
keywordsPanel.setBorder (BorderFactory.createTitledBorder (""));
keywordsButton = new JButton (" Keywords ");
keywordsButton.addActionListener (this);
informalDescButton = new JButton (" Informal Desc ");
informalDescButton.addActionListener (this);
formalDescButton = new JButton (" Formal Desc ");
formalDescButton.addActionListener (this);
keywordsPanel.add (keywordsButton);
keywordsPanel.add (informalDescButton);
keywordsPanel.add (formalDescButton);

okPanel = new JPanel (new GridBagLayout ());
okButton = new JButton ("OK");
okButton.addActionListener (this);
cancelButton = new JButton ("Cancel");
cancelButton.addActionListener (this);
helpButton = new JButton ("Help");
helpButton.addActionListener (this);
gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
okPanel.add (okButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 4;
okPanel.add (cancelButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 7;
okPanel.add (helpButton, gbc);

```

```

        box.add (namePanel);
        box.add (Box.createVerticalStrut (5));
        box.add (triggerPanel);
        box.add (Box.createVerticalStrut (5));
        box.add (timingPanel);
        box.add (Box.createVerticalStrut (2));
        box.add (guardsPanel);
        box.add (Box.createVerticalStrut (2));
        box.add (keywordsPanel);
        box.add (Box.createVerticalStrut (5));
        box.add (okPanel);
        box.add (Box.createVerticalStrut (3));

        getContentPane ().add (box, BorderLayout.CENTER);
    }

    public JComboBox getUnitsCombo ()
    {
        JComboBox c = new JComboBox ();
        c.addItem ("microsec");
        c.addItem ("ms");
        c.addItem ("sec");
        c.addItem ("min");
        c.addItem ("hours");
        return c;
    }
    //SYT
    /**
     * set vertex property
     * @ v - target vertex to set up property
     */
    public void setVertex (Vertex v)
    {
        changeStatus = UNCHANGED;

        targetVertex = v;
        //add 7/23/00 SYT
        currentTimingType = targetVertex.getTimingType();

        tempVertex = (Vertex) v.clone ();

        nameField.setText (v.getLabel ());
        //add 4/1/00 SYT
        networkMappingField.setText(v.getNetWorkLabel());
        //add SYT
        operatorCombo.removeActionListener (this);
        operatorCombo.setSelectedItem ("Operator");
        if (v.isTerminator ())
            operatorCombo.setSelectedItem ("Terminator");
        operatorCombo.addActionListener (this);
        //add 7/4/00 SYT
        if (tempVertex.getCriticalStatus()==1)
        {
            hardRB.setSelected(true);
        }
        else if (tempVertex.getCriticalStatus()==2)
        {
            softRB.setSelected(true);
        }
    }

    if (tempVertex.getImpLanguage ().equalsIgnoreCase ("ada"))
        languageCombo.setSelectedIndex (0);
    else
        languageCombo.setSelectedIndex (1);
    triggerCombo.setSelectedIndex (tempVertex.getTriggerType ());
    // Otherwise it goes into actionperformed
    //close 7/25/00 SYT
    //timingCombo.removeActionListener (this);
    // and deletes the element of the vector
    timingCombo.setSelectedIndex (targetVertex.getTimingType ());

    timingCombo.addActionListener(this);

    resetTimingPanelComponents ();

```



```

ifCondField.setText (v.getIfCondition ());

if (triggerCombo.getSelectedIndex () == Vertex.UNPROTECTED)
    triggerStreamsButton.setEnabled (false);
else
    triggerStreamsButton.setEnabled (true);

PSDLTime met = tempVertex.getMet ();

if (met != null)
{
    metField.setText (String.valueOf (met.getTimeValue ()));
    metUnitsCombo.setSelectedIndex (met.getTimeUnits ());
}
else
{
    metField.setText ("");
    metUnitsCombo.setSelectedIndex (1);
}

if (tempVertex.getTimingType () == Vertex.PERIODIC)
{
    PSDLTime period = tempVertex.getPeriod ();
    PSDLTime fw = tempVertex.getFinishWithin ();
    periodLabel.setText ("Period : ");
    finishWithinLabel.setText ("Finish Within : ");
    if (period != null)
    {
        periodField.setText (String.valueOf (period.getTimeValue ()));
        periodUnitsCombo.setSelectedIndex (period.getTimeUnits ());
    }
    else
    {
        periodField.setText ("");
        periodUnitsCombo.setSelectedIndex (1);
    }
    if (fw != null)
    {
        fwField.setText (String.valueOf (fw.getTimeValue ()));
        fwUnitsCombo.setSelectedIndex (fw.getTimeUnits ());
    }
    else
    {
        fwField.setText ("");
        fwUnitsCombo.setSelectedIndex (1);
    }
}
else if (tempVertex.getTimingType () == Vertex.SPORADIC)
{
    PSDLTime mcp = tempVertex.getMcp ();
    PSDLTime mrt = tempVertex.getMrt ();
    periodLabel.setText ("MCP : ");
    finishWithinLabel.setText ("MRT : ");
    if (mcp != null)
    {
        periodField.setText (String.valueOf (mcp.getTimeValue ()));
        periodUnitsCombo.setSelectedIndex (mcp.getTimeUnits ());
    }
    else
    {
        periodField.setText ("");
        periodUnitsCombo.setSelectedIndex (1);
    }
    if (mrt != null)
    {
        fwField.setText (String.valueOf (mrt.getTimeValue ()));
        fwUnitsCombo.setSelectedIndex (mrt.getTimeUnits ());
    }
    else
    {
        fwField.setText ("");
        fwUnitsCombo.setSelectedIndex (1);
    }
}
else
{
    fwUnitsCombo.setSelectedIndex (1);
    periodUnitsCombo.setSelectedIndex (1);
}

```

```

    }
    //add 7/19/00 SYT
    if(((DefaultMutableTreeNode)v.getParent()).isRoot()
        |((Vertex)(((DefaultMutableTreeNode)v.getParent()))
            .getTimingType() == 0 )
    {
        timingCombo.setEnabled(true);
    }
    else
        timingCombo.setEnabled(false);

    //add 8/22/00 SYT
    if(((DefaultMutableTreeNode)v.getParent()).isRoot())
    {
        //non-time-critical
        if(v.getTimingType() != 0)
        {
            softRB.setEnabled(true);
            hardRB.setEnabled(true);
        }
    }
    else
    {
        softRB.setEnabled(false);
        hardRB.setEnabled(false);
    }
    if (tempVertex.isTerminator ())
    {
        metLabel.setEnabled (false);
        metField.setEnabled (false);
        metUnitsCombo.setEnabled (false);
    }
}
//add 7/25/00 SYT
public JComboBox getTimingCombo()
{
    return timingCombo;
}

//add 7/24/00 SYT
/**
 * pop up a timing type change dialog if necessary
 */
private void popDlg()
{
    //int localTimingType = targetVertex.getTimingType();
    if( ((DefaultMutableTreeNode)(targetVertex.getParent())).isRoot()
        & !targetVertex.isLeaf())
    {
        //not NON-TIME-CRITICAL
        if(timingCombo.getSelectedIndex() != 0 )
        {
            if(currentTimingType != timingCombo.getSelectedIndex() )
            {
                new TimingTypeChangedDlg( this ,1 );

                // currentTimingType = timingCombo.getSelectedIndex();
            }
            else
                resetTiming(false);
        }
        else
        {
            currentTimingType = 0;
            resetTiming(true);
        }
    }
    else
        resetTiming(true);

    if(timingCombo.getSelectedIndex() == 0)
        tempVertex.setTimingType(0);
    else if(timingCombo.getSelectedIndex() ==1)
        tempVertex.setTimingType(1);
    else
        tempVertex.setTimingType(2);
}

```

```

//add 8/19/00 SYT
/**
 * Use when change terminator to operator
 * All operators in direction will be modified to terminator
 */
private void operatorChangedpopDlg()
{
    //int localTimingType = targetVertex.getTimingType();
    if(!tempVertex.isTerminator())
    {
        if( !((DefaultMutableTreeNode)targetVertex).isLeaf() )
            new TimingTypeChangedDlg( this, 2 );
    }
}

//add 8/13/00 SYT
/**
 * synchronize period setup
 */
public void updatePeriod()
{
    //timing type is PERIODIC SYT
    if(targetVertex.getTimingType()== 1)
    {
        PSDLTime period = tempVertex.getPeriod();

        Vertex topestParent = returnTopestParent();
        if(!topestParent.isLeaf())
        {
            for(Enumeration e = topestParent.breadthFirstEnumeration();
                e.hasMoreElements();)
            {
                Object DFC = e.nextElement();
                if(DFC instanceof Vertex)
                {
                    //Vertex localV = (Vertex)e.nextElement();
                    Vertex localV=(Vertex) DFC;
                    localV.setPeriod(period);
                }
            }
        }
    }
}

//add 8/13/00 SYT
/**
 * find the topest parent node
 */
public Vertex returnTopestParent()
{
    Vertex topestParent = targetVertex;
    while(!((DefaultMutableTreeNode)topestParent.getParent()).isRoot() )
    {
        topestParent = (Vertex)(topestParent.getParent());
    }
    return topestParent;
}

// add 7/19/00 SYT
/**
 * synchronize children timing setup
 */
public void updateChildTiming()
{
    Vertex temp;

    //modified 7/23/00 SYT
    if(isTimingTypeChanged)
    {
        for(Enumeration e = targetVertex.breadthFirstEnumeration();
            e.hasMoreElements(); )
        {
            DataFlowComponent DFC = (DataFlowComponent)(e.nextElement());
            if(DFC instanceof Vertex & !DFC.equals(targetVertex))
            {
                temp = (Vertex)DFC;

                //NON_TIME_CRITICAL DEFAULT: do nothing
            }
        }
    }
}

```

```

if (targetVertex.getTimingType() == 0) //NON_TIME_CRITICAL
{
    //Closed 7/28/00 SYT
    //no restriction to sub-component .
    /*
    temp.setCriticalStatus(3);
    temp.setTimingType(targetVertex.getTimingType());

    if (!targetVertex.isTerminator())
    temp.setMet (null);
    if (targetVertex.isTerminator())
    temp.setMetReqmts (targetVertex.getMetReqmts());
    else
    temp.getMetReqmts ().removeAllElements ();

    temp.setPeriod(null);
    temp.getPeriodReqmts().removeAllElements();
    temp.setFinishWithin(null);
    temp.getFinishWithinReqmts().removeAllElements();

    temp.setMcp(null);
    temp.getMcpReqmts().removeAllElements();
    temp.setMrt(null);
    temp.getMrtReqmts().removeAllElements();
    */
}
//PERIODIC DEFAULT: make child has default timing setup
// MET = 0 AND "PERIODIC"
else if (targetVertex.getTimingType() == 1) //PERIODIC
{
    temp.setTimingType(targetVertex.getTimingType());

    //temp.setCriticalStatus(targetVertex.getCriticalStatus());
    temp.setTimingType(targetVertex.getTimingType());

    if (!targetVertex.isTerminator())
    temp.setMet(new PSDLTime());
    /*
    temp.setMetReqmts(targetVertex.getMetReqmts());
    temp.setPeriod(targetVertex.getPeriod());
    temp.setPeriodReqmts(targetVertex.getPeriodReqmts());
    temp.setFinishWithin(targetVertex.getFinishWithin());
    temp.setFinishWithinReqmts(targetVertex.getFinishWithinReqmts());
    */
    temp.setMcp(null);
    temp.getMcpReqmts().removeAllElements();
    temp.setMrt(null);
    temp.getMrtReqmts().removeAllElements();
}
//SPORADIC DEFAULT: make child has default timing setup
// MET = 0 AND "SPORADIC"
else if (targetVertex.getTimingType() == 2) //SPORADIC
{
    temp.setTimingType(targetVertex.getTimingType());

    //temp.setCriticalStatus(targetVertex.getCriticalStatus());
    temp.setTimingType(targetVertex.getTimingType());

    if (!targetVertex.isTerminator())
    temp.setMet(new PSDLTime() );
    /*
    temp.setMetReqmts(targetVertex.getMetReqmts());
    temp.setMcp(targetVertex.getMcp());
    temp.setMcpReqmts(targetVertex.getMcpReqmts());
    temp.setMrt(targetVertex.getMrt());
    temp.setMrtReqmts(targetVertex.getMrtReqmts());
    */
    temp.setPeriod(null);
    temp.getPeriodReqmts().removeAllElements();
    temp.setFinishWithin(null);
    temp.getFinishWithinReqmts().removeAllElements();
}
}
}
}
}

```

```

/**
 * 4/19/00 SYT
 * get this selected vertex
 */
public Vertex getVertex ()
{
    return targetVertex;
}

public void setDisplayVertex (DisplayVertex v)
{
    dVertex = v;
    setVisible (true);
}
//add 8/1/00 SYT
/**
 * reset timing setup
 * @ boolean
 */
public void resetTiming(boolean choose)
{
    //resetTimingPanelComponents ();
    if(choose)
    {
        resetTimingPanelComponents ();
        tempVertex.getMetReqmts ().removeAllElements ();
        tempVertex.getPeriodReqmts ().removeAllElements ();
        tempVertex.getFinishWithinReqmts ().removeAllElements ();
        tempVertex.getMrtReqmts ().removeAllElements ();
        tempVertex.getMcpReqmts ().removeAllElements ();

        if (timingCombo.getSelectedIndex () == Vertex.SPORADIC)
        {
            periodLabel.setText (" MCP :          ");
            finishWithinLabel.setText (" MRT :          ");
        }
        //modified 7/4/00 SYT
        else if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
        {
            periodLabel.setText (" Period : ");
            finishWithinLabel.setText (" Finish within : ");
        }

        if (tempVertex.isTerminator ())
        {
            metLabel.setEnabled (false);
            metField.setEnabled (false);
            metUnitsCombo.setEnabled (false);
        }
    }
    //set property back
    else
    {
        if (targetVertex.getTimingType() == Vertex.SPORADIC)
        {
            periodLabel.setText (" MCP :          ");
            finishWithinLabel.setText (" MRT :          ");
        }
        //modified 7/4/00 SYT
        else if (targetVertex.getTimingType() == Vertex.PERIODIC)
        {
            periodLabel.setText (" Period : ");
            finishWithinLabel.setText (" Finish within : ");
        }
    }
}

//add 8/1/00 SYT
/**
 * reset timing setup
 * @ boolean
 */
public void resetVertexType(boolean choose)
{
    //Operator
    if(choose)
    {

```

```

        //Non-time-critical
        if(tempVertex.getTimingType() == 0)
        {
            metField.enable(false);
            metField.setText("");
            metReqByButton.setEnabled(false);
            metLabel.setEnabled (false);
            metUnitsCombo.setEnabled (false);
            metUnitsCombo.setSelectedIndex (PSDLTime.ms);
            tempVertex.getMetReqmts().removeAllElements();
        }
        else
        {
            metLabel.setEnabled (true);
            metField.enable(true);
            metField.setText("");
            metReqByButton.setEnabled(true);
            metUnitsCombo.setEnabled (true);
        }
    }
    //Terminator
    else
    {
        metField.enable(false);
        metField.setText("0");
        metReqByButton.setEnabled(true);
        metLabel.setEnabled (false);
        metUnitsCombo.setEnabled (false);
        metUnitsCombo.setSelectedIndex (PSDLTime.ms);
    }
}

public void resetTimingPanelComponents ()
{
    metLabel.setEnabled (true);
    if (!tempVertex.isTerminator ())
    {
        metField.setText ("");
        metField.setEnabled (true);
        metUnitsCombo.setEnabled (true);
        metReqByButton.setEnabled (true);
        periodLabel.setEnabled (true);
        periodField.setText ("");
        periodField.setEnabled (true);
        periodUnitsCombo.setEnabled (true);
        periodReqByButton.setEnabled (true);
        finishWithinLabel.setEnabled (true);
        fwField.setText ("");
        fwField.setEnabled (true);
        fwUnitsCombo.setEnabled (true);
        fwReqByButton.setEnabled (true);
        if (timingCombo.getSelectedIndex () == Vertex.NON_TIME_CRITICAL)
        {
            //7/4/00 SYT
            criticalnessLabel.setEnabled(false);
            hardRB.setEnabled(false);
            softRB.setEnabled(false);

            periodLabel.setEnabled (false);
            periodField.setEnabled (false);
            periodUnitsCombo.setEnabled (false);
            periodReqByButton.setEnabled (false);
            finishWithinLabel.setEnabled (false);
            fwField.setEnabled (false);
            fwUnitsCombo.setEnabled (false);
            fwReqByButton.setEnabled (false);
            if (!tempVertex.isTerminator ())
            {
                metLabel.setEnabled (false);
                metField.setEnabled (false);
                metUnitsCombo.setEnabled (false);
                metReqByButton.setEnabled (false);
            }
        }
    }
    else
    {
        criticalnessLabel.setEnabled(true);
        hardRB.setEnabled(true);
    }
}

```

```

        softRB.setEnabled(true);
        setButtonText (triggerReqByButton, tempVertex.getTriggerReqmts ());
        setButtonText (triggerStreamsButton, tempVertex.getTriggerStreamsList ());
        setButtonText (metReqByButton, tempVertex.getMetReqmts ());
        setButtonText (periodReqByButton, tempVertex.getPeriodReqmts ());
        setButtonText (periodReqByButton, tempVertex.getMcpReqmts ());
        setButtonText (fwReqByButton, tempVertex.getFinishWithinReqmts ());
        setButtonText (fwReqByButton, tempVertex.getMrtReqmts ());
        setButtonText (outputGuardsButton, tempVertex.getOutputGuardList ());
        setButtonText (exceptionGuardsButton
                        , tempVertex.getExceptionGuardList ());
        setButtonText (exceptionListButton, tempVertex.getExceptionList ());
        setButtonText (timerOpsButton, tempVertex.getTimerOpList ());
        setButtonText (keywordsButton, tempVertex.getKeywordList ());
        setButtonText (informalDescButton, tempVertex.getInformalDesc ());
        setButtonText (formalDescButton, tempVertex.getFormalDesc ());
    }
}
//add 4/22/00 SYT
public JComboBox getTriggerCombo()
{
    return triggerCombo;
}
public void actionPerformed (ActionEvent e)
{
    if (e.getSource () == ifConditionButton)
    {
        TextEditor.openDialog ("Operator Trigger If Condition"
                                , "View or Edit Operator Trigger If Condition"
                                , ifCondField.getText ()
                                , GrammarCheck.EXPRESSION, true);
        ifCondField.setText (TextEditor.getString ());
        tempVertex.setIfCondition (ifCondField.getText ());
    }
    else if (e.getSource () == triggerReqByButton)
    {
        IdListEditor.openDialog (tempVertex.getTriggerReqmts ());
        tempVertex.setTriggerReqmts (IdListEditor.getIDList ());
        setButtonText (triggerReqByButton, IdListEditor.getIDList ());
    }
    else if (e.getSource () == triggerStreamsButton)
    {
        IdListEditor.openDialog (tempVertex.getTriggerStreamsList ());
        tempVertex.setTriggerStreamsList (IdListEditor.getIDList ());
        setButtonText (triggerStreamsButton, IdListEditor.getIDList ());
    }
    else if (e.getSource () == metReqByButton)
    {
        IdListEditor.openDialog (tempVertex.getMetReqmts ());
        tempVertex.setMetReqmts (IdListEditor.getIDList ());
        setButtonText (metReqByButton, IdListEditor.getIDList ());
    }
    else if (e.getSource () == periodReqByButton)
    {
        if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
        {
            IdListEditor.openDialog (tempVertex.getPeriodReqmts ());
            tempVertex.setPeriodReqmts (IdListEditor.getIDList ());
        }
        else if (timingCombo.getSelectedIndex () == Vertex.SPORADIC)
        {
            IdListEditor.openDialog (tempVertex.getMcpReqmts ());
            tempVertex.setMcpReqmts (IdListEditor.getIDList ());
        }
        setButtonText (periodReqByButton, IdListEditor.getIDList ());
    }
    else if (e.getSource () == fwReqByButton)
    {
        if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
        {
            IdListEditor.openDialog (tempVertex.getFinishWithinReqmts ());
            tempVertex.setFinishWithinReqmts (IdListEditor.getIDList ());
        }
        else if (timingCombo.getSelectedIndex () == Vertex.SPORADIC)
        {
            IdListEditor.openDialog (tempVertex.getMrtReqmts ());
            tempVertex.setMrtReqmts (IdListEditor.getIDList ());
        }
    }
}

```

```

        setButtonText (fwReqByButton, IdListEditor.getIDList ());
    }
    else if (e.getSource () == outputGuardsButton)
    {
        TextEditor.openDialog ("Operator Output Guard"
                               , "View or Edit Operator Output Guard Equation"
                               , tempVertex.getOutputGuardList ()
                               , GrammarCheck.CHECK_OUTPUT_GUARDS
                               , true);
        tempVertex.setOutputGuardList (TextEditor.getString ());
        setButtonText (outputGuardsButton, TextEditor.getString ());
    }
    else if (e.getSource () == exceptionGuardsButton)
    {
        TextEditor.openDialog ("Operator Exceptions"
                               , "View or Edit Operator Exceptions"
                               , tempVertex.getExceptionGuardList ()
                               , GrammarCheck.CHECK_EXCEPTION_GUARDS
                               , true);
        tempVertex.setExceptionGuardList (TextEditor.getString ());
        setButtonText (exceptionGuardsButton, TextEditor.getString ());
    }
    else if (e.getSource () == exceptionListButton)
    {
        TextEditor.openDialog ("Operator Exceptions"
                               , "View or Edit Operator Exceptions"
                               , tempVertex.getExceptionList ()
                               , GrammarCheck.CHECK_EXCEPTION_LIST
                               , true);
        tempVertex.setExceptionList (TextEditor.getString ());
        setButtonText (exceptionListButton, TextEditor.getString ());
    }
    else if (e.getSource () == timerOpsButton)
    {
        TextEditor.openDialog ("Operator Timers"
                               , "View or Edit Operator Timers"
                               , tempVertex.getTimerOpList ()
                               , GrammarCheck.CHECK_TIMER_OPS
                               , true);
        tempVertex.setTimerOpList (TextEditor.getString ());
        setButtonText (timerOpsButton, TextEditor.getString ());
    }
    else if (e.getSource () == keywordsButton)
    {
        IdListEditor.openDialog (tempVertex.getKeywordList ());
        tempVertex.setKeywordList (IdListEditor.getIDList ());
        setButtonText (keywordsButton, IdListEditor.getIDList ());
    }
    else if (e.getSource () == informalDescButton)
    {
        TextEditor.openDialog ("Informal Design Description"
                               , "View or Edit Informal Description"
                               , tempVertex.getInformalDesc ()
                               , GrammarCheck.INFORMAL_DESCRIPTION
                               , true);
        tempVertex.setInformalDesc (TextEditor.getString ());
        setButtonText (informalDescButton, TextEditor.getString ());
    }
    else if (e.getSource () == formalDescButton)
    {
        TextEditor.openDialog ("Formal Design Description"
                               , "View or Edit Formal Description"
                               , tempVertex.getFormalDesc ()
                               , GrammarCheck.FORMAL_DESCRIPTION
                               , true);
        tempVertex.setFormalDesc (TextEditor.getString ());
        setButtonText (formalDescButton, TextEditor.getString ());
    }
    else if (e.getSource () == okButton)
    {
        boolean exceptionOccurred = false;

        String str = nameField.getText ();
        //add 4/1/00 SYT
        String networkStr = networkMappingField.getText();

        /* should call GrammarCheck.OP_ID instead of GrammarCheck.ID

```



```

* if (!GrammarCheck.isValid (str, GrammarCheck.ID)) {
*/

/**
* 11/6/99, 4/19/00 SYT
* if (!GrammarCheck.isValid (str, GrammarCheck.OP_ID)) {
* OP_ID is no longer used.
*/
if (targetVertex.isLeaf())
{
    if (!GrammarCheck.isValid (str, GrammarCheck.OP_ID))
    {
        showErrorDialog ("Illegal vertex name");
        exceptionOccurred = true;
    }
}
else
{
    if (!GrammarCheck.isValid (str, GrammarCheck.ID))
    {
        showErrorDialog("Illegal vertex name.(Type name is not allowed)");
        exceptionOccurred = true;
    }
}
//add network attribute SYT
if (networkStr.length()!=0)
{
    //refere to the def of string_literal in PsdlGrammar
    if ( (networkStr.indexOf('\\ "') != -1 )
        | (networkStr.indexOf('{') != -1) )
    {
        showErrorDialog ("Illegal network_mapping name" );
        exceptionOccurred = true;
    }
}

if (timingCombo.getSelectedIndex () != Vertex.NON_TIME_CRITICAL)
{
    str = metField.getText ();
    //bugs: 8/21/00 SYT
    if(!(tempVertex.isTerminator() ))
    //if (!(targetVertex.isTerminator ()))
    {
        if (!GrammarCheck.isValid (str, GrammarCheck.INTEGER_LITERAL))
        {
            showErrorDialog ("Illegal value for met field");
            exceptionOccurred = true;
        }
    }
    str = periodField.getText ();
    if (str.length () != 0)
    {
        if (!GrammarCheck.isValid (str, GrammarCheck.INTEGER_LITERAL))
        {
            if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
                showErrorDialog ("Illegal value for period field");
            else
                showErrorDialog ("Illegal value for mcp field");
            exceptionOccurred = true;
        }
    }
    str = fwField.getText ();
    if (str.length () != 0)
    {
        if (!GrammarCheck.isValid (str, GrammarCheck.INTEGER_LITERAL))
        {
            if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
                showErrorDialog ("Illegal value for finish within field");
            else
                showErrorDialog ("Illegal value for mrt field");
            exceptionOccurred = true;
        }
    }
}

if (!exceptionOccurred)
{

```

```

targetVertex.setLabel (nameField.getText ());
//add 4/1/00 SYT
targetVertex.setNetWorkLabel(networkMappingField.getText());

targetVertex.setImpLanguage
    ((String) languageCombo.getSelectedItem ());

targetVertex.setTriggerType (triggerCombo.getSelectedIndex ());
if (triggerCombo.getSelectedIndex () != Vertex.UNPROTECTED)
    targetVertex.setTriggerStreamsList
        (tempVertex.getTriggerStreamsList ());
else
    targetVertex.setTriggerStreamsList (new Vector ());

targetVertex.setTimingType (timingCombo.getSelectedIndex ());
//set time critical condition 7/4/00 SYT
if ( timingCombo.getSelectedIndex () != Vertex.NON_TIME_CRITICAL
    & ((DefaultMutableTreeNode) (targetVertex.getParent()) ).isRoot())
{
    if(hardRB.isSelected())
    {
        for(Enumeration en =targetVertex.breadthFirstEnumeration()
            ;en.hasMoreElements(); )
        {
            Object ob=en.nextElement();
            if(ob instanceof Vertex)
                ((Vertex)ob).setCriticalStatus(1);
        }
    }
    else
    {
        for(Enumeration en =targetVertex.breadthFirstEnumeration()
            ;en.hasMoreElements(); )
        {
            Object ob=en.nextElement();
            if(ob instanceof Vertex)
                ((Vertex)ob).setCriticalStatus(2);
        }
    }
}
else
{
    for(Enumeration en =targetVertex.breadthFirstEnumeration()
        ;en.hasMoreElements(); )
    {
        Object ob=en.nextElement();
        if(ob instanceof Vertex)
            ((Vertex)ob).setCriticalStatus(3);
    }
}
//corrected 7/22/00 SYT
// if (timingCombo.getSelectedIndex () != Vertex.NON_TIME_CRITICAL
//     && !(targetVertex.isTerminator ()))

if ( timingCombo.getSelectedIndex () != Vertex.NON_TIME_CRITICAL )
{
    targetVertex.setMet (new PSDLTime (Integer.parseInt
        (metField.getText ())
        , metUnitsCombo.getSelectedIndex ());
    targetVertex.setMetReqmts (tempVertex.getMetReqmts ());
}
//bugs: 8/21/00 SYT
//else if (!(targetVertex.isTerminator ()))
else if (!(tempVertex.isTerminator ()))
{
    targetVertex.setMet (null);
    targetVertex.getMetReqmts ().removeAllElements ();
}
//add 7/22/00 SYT
else
{
    //bug 8/3/00 SYT
    //targetVertex.setMet (null);
    targetVertex.setMet (new PSDLTime
        ());
    targetVertex.setMetReqmts (tempVertex.getMetReqmts ());
}

```

```

}

if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
{
    if (periodField.getText ().length () != 0)
    {
        targetVertex.setPeriod (new PSDLTime (Integer.parseInt
            (periodField.getText ())
            , periodUnitsCombo.getSelectedIndex ());

        targetVertex.setPeriodReqmts (tempVertex.getPeriodReqmts ());
    }
    else
    {
        targetVertex.setPeriod (null);
        targetVertex.getPeriodReqmts ().removeAllElements ();
    }
    if (fwField.getText ().length () != 0)
    {
        targetVertex.setFinishWithin (new PSDLTime
            (Integer.parseInt (fwField.getText ())
            , fwUnitsCombo.getSelectedIndex ());
        targetVertex.setFinishWithinReqmts
            (tempVertex.getFinishWithinReqmts ());
    }
    else
    {
        targetVertex.setFinishWithin (null);
        targetVertex.getFinishWithinReqmts ().removeAllElements ();
    }
    targetVertex.getMcpReqmts ().removeAllElements ();
    targetVertex.getMrtReqmts ().removeAllElements ();
}
else if (timingCombo.getSelectedIndex () == Vertex.SPORADIC)
{
    if (periodField.getText ().length () != 0)
    {
        targetVertex.setMcp (new PSDLTime
            (Integer.parseInt (periodField.getText ())
            , periodUnitsCombo.getSelectedIndex ());

        targetVertex.setMcpReqmts (tempVertex.getMcpReqmts ());
    }
    else
    {
        targetVertex.setMcp (null);
        targetVertex.getMcpReqmts ().removeAllElements ();
    }
    if (fwField.getText ().length () != 0)
    {
        targetVertex.setMrt (new PSDLTime
            (Integer.parseInt (fwField.getText ())
            , fwUnitsCombo.getSelectedIndex ());

        targetVertex.setMrtReqmts (tempVertex.getMrtReqmts ());
    }
    else
    {
        targetVertex.setMrt (null);
        targetVertex.getMrtReqmts ().removeAllElements ();
    }
    targetVertex.getPeriodReqmts ().removeAllElements ();
    targetVertex.getFinishWithinReqmts ().removeAllElements ();
}

targetVertex.setIfCondition (tempVertex.getIfCondition ());
targetVertex.setOutputGuardList (tempVertex.getOutputGuardList ());
targetVertex.setExceptionGuardList
    (tempVertex.getExceptionGuardList ());
targetVertex.setExceptionList (tempVertex.getExceptionList ());
targetVertex.setTimerOpList (tempVertex.getTimerOpList ());
targetVertex.setInformalDesc (tempVertex.getInformalDesc ());
targetVertex.setFormalDesc (tempVertex.getFormalDesc ());
targetVertex.setTriggerReqmts (tempVertex.getTriggerReqmts ());
targetVertex.setKeywordList (tempVertex.getKeywordList ());

dVertex.setLabelShape ((Graphics2D) parentFrame

```

```

                                .getDrawPanel ().getGraphics ());
dVertex.setMetShape ((Graphics2D) parentFrame
                                .getDrawPanel ().getGraphics ());
//add 4/22/00 SYT
setVisible(false);
if( ( triggerCombo.getSelectedIndex() == Vertex.UNPROTECTED) &&
    !( targetVertex.getTriggerStreamsList().isEmpty() ) )
{
    targetVertex.getTriggerStreamsList().removeAllElements();
}
if( (triggerCombo.getSelectedIndex () != Vertex.UNPROTECTED)
    && (targetVertex.getTriggerStreamsList().isEmpty() ) )
{
    showErrorDialog("You are supposed to set up Trigger Stream List."
        + "\n"
        + "or set Trigger as \"Unprotected\"");
    setVisible(true);
}
if (changeStatus == TO_OPERATOR)
{
    targetVertex.setTerminator (false);
    parentFrame.getDrawPanel ().changeLevel ((Vertex) targetVertex
                                                .getParent ());
}
else if (changeStatus == TO_TERMINATOR)
{
    DataFlowComponent dfc;
    for (Enumeration enum = targetVertex.breadthFirstEnumeration ()
        ; enum.hasMoreElements ();)
    {
        dfc = (DataFlowComponent) enum.nextElement ();
        if (dfc instanceof Vertex)
        {
            ((Vertex) dfc).setTerminator (true);
            ((Vertex) dfc).setMet (new PSDLTime (0, PSDLTime.ms));
        }
    }
    parentFrame.getDrawPanel ().changeLevel ((Vertex) targetVertex
                                                .getParent ());
}
else
{
    parentFrame.getDrawPanel ().clearAllComponentsFromScreen (null);
    parentFrame.getDrawPanel ().paint (parentFrame.getDrawPanel ()
        .getGraphics ());
}
parentFrame.getTreePanel ().repaint ();
parentFrame.setSaveRequired (true);

//add 7/19/00
//update timing of the children node
updateChildTiming();
//add 8/13/00 update period
updatePeriod();
}

}
else if (e.getSource () == cancelButton)
{
    setVisible (false);
}
else if (e.getSource () == helpButton)
{
}
/* bugs and has been modified 8/19/00 SYT */
else if (e.getSource () == operatorCombo)
{
    changeStatus = operatorCombo.getSelectedIndex ();
    if (changeStatus == TO_TERMINATOR)
    {
        //add 8/19/00 SYT
        operatorCombo.hidePopup();
        operatorChangedpopDlg();
        if(isVertexTypeChanged | tempVertex.isLeaf())
        {
            operatorCombo.setSelectedIndex(1);
            tempVertex.setTerminator(true);
            resetVertexType(false);
        }
    }
}

```

```

        isVertexTypeChanged = false;
    }
    else
        operatorCombo.setSelectedIndex(0);
}
else if (changeStatus == TO_OPERATOR)
{
    //add 8/19/00 SYT
    tempVertex.setTerminator(false);
    resetVertexType(true);
    /* bugs: 8/19/00 SYT */
    /*
    metField.setEnabled (true);
    metLabel.setEnabled (true);
    metUnitsCombo.setEnabled (true);
    if (targetVertex.getMet () != null)
    {
        metField.setText (String.valueOf (targetVertex.getMet ()
                                           .getTimeValue ()));
        metUnitsCombo.setSelectedIndex (targetVertex.getMet ()
                                         .getTimeUnits ());
        metField.setText ("");
        metUnitsCombo.setSelectedIndex (PSDLTime.ms);
    }
    else
    {
        metField.setText ("");
        metUnitsCombo.setSelectedIndex (PSDLTime.ms);
    }*/

    }
    /* if (targetVertex.isTerminator () && changeStatus == TO_TERMINATOR
    || !targetVertex.isTerminator () && changeStatus == TO_OPERATOR)

        changeStatus = UNCHANGED;*/
}
else if (e.getSource () == triggerCombo)
{
    if (triggerCombo.getSelectedIndex () == Vertex.UNPROTECTED)
        triggerStreamsButton.setEnabled (false);
    else
        triggerStreamsButton.setEnabled (true);
}
/* Bugs: closed 7/25/00 SYT
else if (e.getSource () == timingCombo)
{
    resetTimingPanelComponents ();
    tempVertex.getMetReqmts ().removeAllElements ();
    tempVertex.getPeriodReqmts ().removeAllElements ();
    tempVertex.getFinishWithinReqmts ().removeAllElements ();
    tempVertex.getMrtReqmts ().removeAllElements ();
    tempVertex.getMcpReqmts ().removeAllElements ();

    if (timingCombo.getSelectedIndex () == Vertex.SPORADIC)
    {
        periodLabel.setText (" MCP :          ");
        finishWithinLabel.setText (" MRT :          ");
    }
    //modified 7/4/00 SYT
    else if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
    {
        periodLabel.setText (" Period : ");
        finishWithinLabel.setText (" Finish within : ");
    }
    if (tempVertex.isTerminator ())
    {
        metLabel.setEnabled (false);
        metField.setEnabled (false);
        metUnitsCombo.setEnabled (false);
    }
}*/
}

public void showErrorDialog (String str)
{
    JOptionPane.showMessageDialog (this, str, "Error Message"

```

```

        , JOptionPane.ERROR_MESSAGE);
    }

    public void setButtonText (JButton b, Object o)
    {
        if ((o instanceof Vector) && (((Vector) o).size () != 0)) ||
            ((o instanceof String) && (((String) o).length () != 0)))
        {
            if (!b.getText ().endsWith ("..."))
                b.setText (b.getText ().trim () + " ...");
        }
        else
        {
            if (b.getText ().endsWith ("..."))
                b.setText (" " + b.getText ().substring (0, b.getText ().
                                                            .length () - 4) + " ");
        }
    }
    //add 7/24/00
    void timingCombo_actionPerformed(ActionEvent e)
    {
        timingCombo.hidePopup();
        popDlg();
    }

} // End of the class VertexProperties
/**
 * modification for new grammar.
 * modify CreatePsdl to make a correct output grammar textfile
 * this file is a driver to mape the phototype to textfile
 * in this class, all methods has been modified for it can judge two
 * type of input stream and can generate proper grammar for output files.
 * @author Shen-Yi Tao
 * @version 1.1
 */
package caps.Parser;

import java.awt.Point;
import java.io.StringReader;
import java.io.StringWriter;
import caps.Psdl.*;
import java.util.Enumeration;
import java.util.Vector;
/**
 * 11/16/99 SYT
 * add
 */
import caps.GraphEditor.Editor;

public class CreatePsdl
{
    // creating a writer for all Vertexs. 11/6/99 SYT
    static StringWriter writer;

    // 11/6/99 SYT
    // writer = new StringWriter ();
    public static void ReInit (StringWriter r)
    {
        writer=r;
    }

    public static String getPsd1 ()
    {
        return writer.toString ();
    }

    public static void build (Vertex root, DataTypes types)
    {
        DataFlowComponent dfc;
        //defined data types 8/9/00 SYT
        DataTypes (types);

        // 11/19/99 SYT
        //boolean firstTimeGetIn =true;

```

```

//map each operator to file. 11/8/99 SYT
for (Enumeration enum = root.breadthFirstEnumeration ()
    ; enum.hasMoreElements ();)
{
    dfc = (DataFlowComponent) enum.nextElement ();

    // 11/19/99 SYT
    //if( firstTimeGetIn )
    // {
    //     writeTitle((Vertex) dfc );
    //     firstTimeGetIn = false;
    // }

    // if this is an operator , output to file. 11/8/99 SYT
    if (dfc instanceof Vertex && !(dfc instanceof External))
        operator ((Vertex) dfc);
}

/**
 * 11/16/99 SYT
 * get phototype name.
 */
public static String getPhototypeName ()
{
    return Editor.prototypeName;
}

public static void dataTypes (DataTypes types)
{
    writer.write (types.toString ());
}

/**
 * 11/16/99 SYT
 * create an output file.
 */
public static void operator (Vertex v)
{
    //writePhototypeName (v);
    operatorSpecification (v);

    operatorImplementation (v);
}

//write specification for operator. 11/8/99 SYT
public static void operatorSpecification (Vertex v)
{
    writer.write ( v.getSpecification (true) );
}

// 11/10/99 SYT
//write implementation for operatot.
//modification for a correct grammar
//need judge two type and generate two text for each.

public static void operatorImplementation (Vertex v)
{
    if (v.isLeaf ())
    {
        if (GrammarCheck.isValid (v.getLabel(), GrammarCheck.ID))
        {
            //change the formate. 5/1/00 SYT
            writer.write ("      IMPLEMENTATION " + v.getImpLanguage ()
                + " " + v.getLabel () + "\n");
            writer.write ("      END\n\n");
        }
        else
        {
            String tempString=v.getLabel();
            String vertexString = "";
            String newString;
            int index = tempString.indexOf(".");
            //change the formate. 5/1/00 SYT
            newString= vertexString.concat("      IMPLEMENTATION "
                + v.getImpLanguage () + " "
                + v.getLabel().substring(0,index) + "\n");
        }
    }
}

```

```

        writer.write (newString);
        writer.write ("        END\n\n");
    }

}
else
{
    //change output format 5/1/00 SYT
    writer.write ("        IMPLEMENTATION\n");
    psdlImplementation (v);
    writer.write ("        END\n\n");
}
}

public static void psdlImplementation (Vertex v)
{
    dataFlowDiagram (v);

    streams (v);

    timers (v);

    controlConstraints (v);

    informalDesc (v);
}

public static void dataFlowDiagram (Vertex v)
{
    writer.write ("        GRAPH\n");
    DataFlowComponent d;
    for (Enumeration enum = v.children (); enum.hasMoreElements ());
    {
        d = (DataFlowComponent) enum.nextElement ();
        if (d instanceof Vertex && !(d instanceof External))
            vertex ((Vertex) d);
    }
    for (Enumeration enum = v.children (); enum.hasMoreElements ());
    {
        d = (DataFlowComponent) enum.nextElement ();
        if (d instanceof Edge)
            edge ((Edge) d);
    }
}

/**
 * 11/6/99 SYT
 * modify vertex(Vertex) to make it able to judge vertex type,
 * and write suitable grammar for it. */

public static void vertex (Vertex v)
{
    // input type : "operator1" 11/9/99 SYT
    if ( GrammarCheck.isValid (v.getLabel(), GrammarCheck.ID))
    {
        writer.write ("        VERTEX " + v.getLabel () + "_" + v.getId ()
            + "_" + (v.getId () -1 ) + " ");
    }
    // input type : "machine.process(inputstream|outputstream)" 11/9/99 SYT
    else
    {
        String tempString=v.getLabel();
        String vertexString = "";
        String newString;
        // find input stream "(" ; 11/9/99 SYT
        int index = tempString.indexOf("(");

        if( index == -1)
        {
            newString = vertexString.concat("        VERTEX " +tempString
                + "_" +v.getId());
        }
        else
        {
            newString = vertexString.concat("        VERTEX "
                +tempString.substring(0,index)
                + "_" +v.getId()+tempString.substring(index));
        }
    }
}

```



```

    }
    writer.write (newString);
}

if (v.getMet () != null)
    writer.write (": " + v.getMet ().toString ());
writer.write ("\n");
vertexProperties (v);
}

public static void vertexProperties (Vertex v)
{
    writer.write ("        PROPERTY x = " + v.getX () + "\n");
    writer.write ("        PROPERTY y = " + v.getY () + "\n");
    writer.write ("        PROPERTY radius = " + (v.getWidth () / 2)
        + "\n");
    writer.write ("        PROPERTY color = " + v.getColor () + "\n");
    writer.write ("        PROPERTY label_font = "
        + v.getLabelFontIndex () + "\n");
    writer.write ("        PROPERTY label_x_offset = "
        + v.getLabelXOffset () + "\n");
    writer.write ("        PROPERTY label_y_offset = "
        + v.getLabelYOffset () + "\n");
    writer.write ("        PROPERTY met_font = "
        + v.getMetFontIndex () + "\n");
    writer.write ("        PROPERTY met_unit = "
        + ((v.getMet () == null) ? 1 : v.getMet ()
            .getTimeUnits ()) + "\n");
    writer.write ("        PROPERTY met_x_offset = " + v.getMetXOffset ()
        + "\n");
    writer.write ("        PROPERTY met_y_offset = " + v.getMetYOffset ()
        + "\n");
    writer.write ("        PROPERTY is_terminator = " + v.isTerminator ()
        + "\n");

    //add 4/1/00 SYT
    if (v.getNetWorkLabel().length()==0)
        writer.write ("        PROPERTY network_mapping = " + "\"\" + "\"\"
            + "\n");
    else
        writer.write ("        PROPERTY network_mapping = " + "\"\"
            + v.getNetWorkLabel() + "\"\" + "\n");

    //add 7/4/00 SYT
    if (v.getCriticalStatus()==1)
        writer.write ("        PROPERTY criticalness = " + "\"\" + \"hard\" + "\"\"
            + "\n" );
    else if (v.getCriticalStatus()==2)
        writer.write ("        PROPERTY criticalness = " + "\"\" + \"soft\" + "\"\"
            + "\n" );
    else
        writer.write ("        PROPERTY criticalness = " + "\"\" + \"none\" + "\"\"
            + "\n" );
}

/**
 * 11/6/99 SYT
 * modify edge(Edge) to make it able to judge vertex type
 * and write suitable grammar for it.
 */
public static void edge (Edge e)
{
    writer.write ("        EDGE " + e.getLabel () + " ");
    if (e.getMet () != null)
        writer.write (": " + e.getMet ().toString () + " ");
    //chane for grammar check error 2/4/00 SYT
    if (e.getSource () instanceof External)
    {
        writer.write (e.getSource ().getLabel ());
        //add 2/4/00 SYT
        writer.write (" -> ");
    }
    else
    {
        /* Debug NullPointerException SYT */
        if (e.getSource() != null)
        {
            if (GrammarCheck.isValid (e.getSource().getLabel(), GrammarCheck.ID))

```

```

        {
            writer.write (e.getSource ().getLabel () + "_"
            + e.getSource ().getId () + "_" + (e.getSource ().getId() -1) );
            writer.write (" -> ");
        }
        else
        {
            String tempString=e.getSource().getLabel();
            //change String to StringBuffer 11/22/99 SYT
            StringBuffer vertexString = new StringBuffer();
            int index = tempString.indexOf("(");
            // make check 11/28/99 SYT
            if( index == -1)
            {
                vertexString.append(tempString);
                vertexString.append("_"
                + Integer.toString(e.getSource().getId()));
            }
            else
            {
                vertexString.append(tempString.substring(0,index));
                vertexString.append("_"
                + Integer.toString(e.getSource().getId()));
                vertexString.append(tempString.substring(index));
            }
            writer.write (vertexString.toString());
            writer.write (" -> ");
        }
    }
}

if (e.getDestination () instanceof External)
    writer.write (e.getDestination ().getLabel ());
else
{
    /* debug NullPointerException SYT */
    if(e.getDestination() != null)
    {
        if (GrammarCheck.isValid (e.getDestination().getLabel()
        , GrammarCheck.ID))
        {
            writer.write (e.getDestination().getLabel ()
            + "_" + e.getDestination().getId () + "_"
            + (e.getDestination().getId() -1 ) );
        }
        else
        {
            String tempString=e.getDestination().getLabel();
            //use StringBuffer instand of Sting to correct
            //"no concate error" 11/22/99 SYT
            StringBuffer vertexString = new StringBuffer();
            int index = tempString.indexOf("(");
            // add 11/28/99 SYT
            if( index == -1)
            {
                vertexString.append(tempString);
                vertexString.append("_" +
                Integer.toString(e.getDestination().getId())+"\\n");
            }
            else
            {
                vertexString.append(tempString.substring(0,index));
                vertexString.append("_" + Integer.toString(e.getDestination().getId()));
                vertexString.append(tempString.substring(index)+"\\n");
            }
        }
        writer.write (vertexString.toString());
    }
}
}
//move 2/15/00 SYT
writer.write ("\\n");
edgeProperties (e);
}

```

```

public static void edgeProperties (Edge e)
{
    Vector points = e.getPoints ();
    Point p = (Point) points.elementAt (points.size () / 2);
    writer.write ("          PROPERTY id = " + e.getId () + "\n");
    writer.write ("          PROPERTY label_font = " + e.getLabelFontIndex () +
"\n");
    writer.write ("          PROPERTY label_x_offset = " + e.getLabelXOffset () +
"\n");
    writer.write ("          PROPERTY label_y_offset = " + e.getLabelYOffset () +
"\n");
    writer.write ("          PROPERTY latency_font = " + e.getMetFontIndex () +
"\n");
    writer.write ("          PROPERTY latency_unit = " +
((e.getMet () == null) ? 1 : e.getMet ().getTimeUnits ()) + "\n");
    writer.write ("          PROPERTY latency_x_offset = " + e.getMetXOffset () +
"\n");
    writer.write ("          PROPERTY latency_y_offset = " + e.getMetYOffset () +
"\n");
    writer.write ("          PROPERTY spline = \"");
    if (e.getSource () instanceof External)
    {
        p = (Point) points.firstElement ();
        writer.write (p.x + " " + p.y + " ");
    }
    for (Enumeration enum = points.elements (); enum.hasMoreElements ();)
    {
        p = (Point) enum.nextElement ();
        if (!p.equals (points.firstElement ()) && !p.equals (points.lastElement ()))
        { // do nothing
            if (!p.equals (points.firstElement ()))
                p = (Point) enum.nextElement ();
            if (!p.equals (points.lastElement ()))
                writer.write (p.x + " " + p.y + " ");
        }
    }
    if (e.getDestination () instanceof External)
    {
        p = (Point) points.lastElement ();
        writer.write (p.x + " " + p.y + " ");
    }
    writer.write ("\"\\n");
}
// data stream modified 5/24/00 SYT
public static void streams (Vertex v)
{
    DataFlowComponent d;
    String str = "";
    for (Enumeration enum = v.children (); enum.hasMoreElements ();)
    {
        d = (DataFlowComponent) enum.nextElement ();
        if (d instanceof Edge)
        {
            Edge e = (Edge) d;
            if (str.lastIndexOf (" " + e.getLabel () + " ") == -1)
            {
                //if stream not found in the parent's node
                //and it is not a state stream SYT
                if (!e.isStateStream ())
                {
                    boolean found = false;
                    // inEdge SYT
                    for (Enumeration in = v.getInEdgesVector().elements();
                        in.hasMoreElements(); )
                    {
                        DataFlowComponent i = (DataFlowComponent) in.nextElement ();
                        if (i instanceof Edge)
                        {
                            if ( ((Edge)i).getLabel().equals(e.getLabel()))
                            {
                                found=true;
                            }
                        }
                    }
                    //outEdge SYT
                    if (!found)
                    {

```

```

        for(Enumeration out = v.getOutEdgesVector().elements();
           out.hasMoreElements();)
        {
            DataFlowComponent o = (DataFlowComponent) out.nextElement
();
            if (o instanceof Edge)
            {
                if( ((Edge)o).getLabel().equals(e.getLabel()))
                {
                    found=true;
                }
            }
        }
        if (!found)
        {
            if(str.length()==0)
                str = str.concat ("          " + e.getLabel () + " : " +
e.getStreamType ());
            else
                str = str.concat (",\n          " + e.getLabel () + " : " +
e.getStreamType ());
        }
        found=false;
    }

    // 5/24/00 SYT
    // else if (!e.isStateStream ())
    // str = str.concat (",\n          " + e.getLabel () + " : " +
e.getStreamType ());
    }
}
if (str.length () != 0)
{
    writer.write ("      DATA STREAM\n");
    writer.write (str + "\n");
}

}

public static void timers (Vertex v)
{
    if (v.getTimerList ().size () != 0)
        writer.write ("      TIMER " + v.extractList (v.getTimerList ()) + "\n");
}

public static void controlConstraints (Vertex v)
{
    writer.write ("      CONTROL CONSTRAINTS\n");
    DataFlowComponent d;
    for (Enumeration enum = v.children (); enum.hasMoreElements ());
    {
        d = (DataFlowComponent) enum.nextElement ();
        if (d instanceof Vertex && !(d instanceof External))
            constraint ((Vertex) d);
    }
}

/**
 * 11/6/99 SYT
 * 1.make it able to judge two type of
 * input stream
 * 2.write constraint
 */
public static void constraint (Vertex v)
{
    // id or op_id 11/16/99 SYT
    if (GrammarCheck.isValid (v.getLabel(), GrammarCheck.ID))
    {
        writer.write ("      OPERATOR " + v.getLabel () + "_" + v.getId () + "_" +
(v.getId () -1 ) + "\n");
    }
    else
    {
        String tempString=v.getLabel();

```

```

        String vertexString = "";
        String newString;
        int index = tempString.indexOf("(");
        // make check 11/28/99 SYT
        if( index == -1)
        {
            newString= vertexString.concat("          OPERATOR " +tempString
            + "_" + v.getId() + "\n");
        }
        else
        {
            newString= vertexString.concat("          OPERATOR "
+tempString.substring(0,index)
            + "_" + v.getId()+tempString.substring(index)+"\n");
        }
        writer.write (newString);

    }

    trigger (v);
    period (v);
    finishWithin (v);
    mcp (v);
    mrt (v);
    outputGuards (v);
    exceptionGuards (v);
    timerOps (v);
}

//modified for correction 4/21/00 SYT
public static void trigger (Vertex v)
{
    if ( ((v.getIfCondition().length() != 0)
    | (v.getTriggerType () != Vertex.UNPROTECTED))
    |v.getTriggerReqmts ().size () != 0 )
    {
        writer.write( "          TRIGGERED " );
        if( v.getTriggerType () != Vertex.UNPROTECTED )
        {
            writer.write ("BY ");
            if (v.getTriggerType () == Vertex.BY_SOME)
                writer.write ("SOME ");
            else
                writer.write ("ALL ");
            writer.write (v.extractList (v.getTriggerStreamsList ()) + "\n" );
        }
        //add 8/9/00 SYT
        else
            writer.write ("\n");

        if(v.getIfCondition().length() != 0)
            writer.write("          IF "+ v.getIfCondition() + "\n");

        if (v.getTriggerReqmts ().size () != 0)
            writer.write ("          REQUIRED BY "
                + v.extractList (v.getTriggerReqmts ())+ "\n");

    }

}

//modified 7/4/00 SYT
public static void period (Vertex v)
{
    if (v.getTimingType() != 0 && v.getPeriod () != null)
    {
        writer.write ("          PERIOD " + v.getPeriod ().toString () + "\n");
        if (v.getPeriodReqmts ().size () != 0)
            writer.write ("          REQUIRED BY " + v.extractList
(v.getPeriodReqmts ()) + "\n");
    }

}

//modified 7/4/00 SYT
public static void finishWithin (Vertex v)
{
    if (v.getTimingType() != 0 && v.getFinishWithin () != null)
    {

```

```

        writer.write ("                FINISH WITHIN " + v.getFinishWithin ().toString ()
+ "\n");
        if (v.getFinishWithinReqmts ().size () != 0)
            writer.write ("                REQUIRED BY " + v.extractList
(v.getFinishWithinReqmts ()) + "\n");
    }

    public static void mcp (Vertex v)
    {
        if (v.getMcp () != null)
        {
            writer.write ("                MINIMUM CALLING PERIOD " + v.getMcp ().toString ()
+ "\n");
            if (v.getMcpReqmts ().size () != 0)
                writer.write ("                REQUIRED BY " + v.extractList (v.getMcpReqmts
()) + "\n");
        }
    }

    public static void mrt (Vertex v)
    {
        if (v.getMrt () != null)
        {
            writer.write ("                MAXIMUM RESPONSE TIME " + v.getMrt ().toString ()
+ "\n");
            if (v.getMrtReqmts ().size () != 0)
                writer.write ("                REQUIRED BY " + v.extractList (v.getMrtReqmts
()) + "\n");
        }
    }

    public static void outputGuards (Vertex v)
    {
        if (v.getOutputGuardList ().length () != 0)
        {
            String str = v.getOutputGuardList ();
            writer.write (v.extractString (str, true));
        }
    }

    }

    public static void exceptionGuards (Vertex v)
    {
        if (v.getExceptionGuardList ().length () != 0)
        {
            String str = v.getExceptionGuardList ();
            writer.write (v.extractString (str, true));
        }
    }

    }

    public static void timerOps (Vertex v)
    {
        if (v.getTimerOpList ().length () != 0)
        {
            String str = v.getTimerOpList ();
            writer.write (v.extractString (str, true));
        }
    }

    }

    public static void informalDesc (Vertex v)
    {
        if (v.getGraphDesc ().length () != 0)
        {
            String str = v.getGraphDesc ();
            writer.write (v.extractString (str, false));
        }
    }

    }

```

```

} // End of the class CreatePsdl
// OCT.14.99 SYT
// add following code for testing Token.kind ; return Token.kind =0 when
// inside PSDL user menu ,select an operator ,pull up property menu,change
// operator mane the current Token.kind=0.
// get concluded : if (temp_token.kind != PsdlParserConstants.EOF)
// return flag = true

```

```

package caps.Parser;

```

```

import java.io.*;
//add 5/28/00 SYT
import caps.Psdl.*;
import java.util.Enumeration ;

```

```

public class GrammarCheck
{
    public static StringReader reader;

    public static final int OP_ID = 1;

    public static final int TYPE_NAME = 2;

    public static final int INTEGER_LITERAL = 3;

    public static final int INITIAL_EXPRESSION = 4;

    public static final int EXPRESSION = 5;

    public static final int CHECK_OUTPUT_GUARDS = 6;

    public static final int CHECK_EXCEPTION_GUARDS = 7;

    public static final int CHECK_EXCEPTION_LIST = 8;

    public static final int CHECK_TIMER_OPS = 9;

    public static final int INFORMAL_DESCRIPTION = 10;

    public static final int FORMAL_DESCRIPTION = 11;

    public static final int DATA_TYPE = 12;

    public static final int CHECK_PARENT_SPEC = 13;

    public static final int ID = 14;

    // * 11/6/99 SYT
    /**
     * delete OP_ID which is no longer used.
     * public static final int OP_ID = 14;
     */
    public static boolean isValid (String str, int kind)
    {
        reader = new StringReader (str);
        PsdlParser.ReInit (reader);
        boolean flag = true;
        try
        {
            switch (kind)
            {
                case OP_ID :
                    PsdlParser.OP_ID();
                    break;

                case TYPE_NAME :
                    PsdlParser.type_name ();
                    break;

                case INTEGER_LITERAL :
                    PsdlParser.integer_literal ();
                    break;

                case INITIAL_EXPRESSION :
                    PsdlParser.initial_expression ();
                    break;

                case EXPRESSION :

```

```

        PsdlParser.expression ();
        break;

    case CHECK_OUTPUT_GUARDS :
        PsdlParser.check_output_guards ();
        break;
    case CHECK_EXCEPTION_GUARDS :
        PsdlParser.check_exception_guards ();
        break;
    case CHECK_EXCEPTION_LIST :
        PsdlParser.check_exception_list ();
        break;
    case CHECK_TIMER_OPS :
        PsdlParser.check_timer_ops ();
        break;
    case INFORMAL_DESCRIPTION :
        PsdlParser.informal_desc ();
        break;
    case FORMAL_DESCRIPTION :
        PsdlParser.formal_desc ();
        break;
    case DATA_TYPE :
        PsdlParser.psdl ();
        break;
    case CHECK_PARENT_SPEC :
        PsdlParser.check_parent_spec ();
        break;
    // this is needed to judge user's input
    case ID :
        PsdlParser.ID();
        break;

    /*
        11/6/99 SYT
        delete check OP_ID
        case OP_ID :
        PsdlParser.op_id();
        break;
    */

    default :
        break;
}

if (PsdlParser.getNextToken ().kind != PsdlParserConstants.EOF) // If there
is not only one id
{
    // System.out.println ("Characters encountered after a valid token");
    // System.out.println ("problem found in GrammarCheck.IsValid");
    flag = false;
}
}
catch (ParseException e)
{
    System.out.println ("Parse exception occurred");
    System.out.println (e);
    flag = false;
}
catch (TokenMgrError e)
{
    System.out.println ("Lexical error occurred");
    System.out.println (e);
    flag = false;
}
catch (Exception e)
{
    System.out.println ("An error occurred during parsing the structure");
    System.out.println (e);
    flag = false;
}
return flag;
}

}

} //end GrammarCheck.java
package caps.Parser;

import javax.swing.tree.DefaultMutableTreeNode;

```



```

import caps.Psdl.*;
import java.util.*;
import caps.GraphEditor.* ;
/**
 * check stream consistence : Local or Global
 * @ author Shen-Yi
 * @ version 1.0
 */
public class StreamConsistenceCheck
{
    public StreamConsistenceCheck()
    {
    }
    // 8/24/00 SYT
    /**
     * check the stream has the same type in PSDL project
     * @ root: the root of this tree
     * @ targetstatus: the local stream to be examined. if null all stream to be
     examined.
     */

    public static void checkConsistence (DefaultMutableTreeNode treeRoot , Edge stream)
    {
        DefaultMutableTreeNode root = treeRoot;
        Edge targetStream = stream;
        Vector globalVector = new Vector();

        //do global
        if (targetStream == null)
        {
            for(Enumeration e = root.breadthFirstEnumeration(); e.hasMoreElements(); )
            {
                DefaultMutableTreeNode temp = (DefaultMutableTreeNode)e.nextElement();
                if(temp instanceof Edge)
                {
                    doLocal(root, (Edge)temp);
                }
            }
        }
        //do local
        else
        {
            doLocal(root, targetStream);
        }
    }

    private static void doLocal(DefaultMutableTreeNode treeRoot ,Edge target)
    {
        DefaultMutableTreeNode root = treeRoot ;
        Edge targetStream = target;
        Vector localVector = new Vector();
        localVector.add(targetStream);

        if(targetStream instanceof Edge)
        {
            String streamName = targetStream.getLabel();

            for(Enumeration e = root.breadthFirstEnumeration(); e.hasMoreElements(); )
            {
                DefaultMutableTreeNode temp = (DefaultMutableTreeNode)e.nextElement();
                if(temp instanceof Edge)
                {
                    if( streamName.equals( ((Edge)temp).getLabel() ) )
                    {
                        String streamType = targetStream.getStreamType();
                        boolean isState = targetStream.isStateStream();
                        String initialVal = targetStream.getInitialValue();
                        PSDLTime lantency = targetStream.getMet();

                        boolean testFlag = false;

                        if(streamType != null)
                        {
                            if( !streamType.equals(((Edge)temp).getStreamType() ) )
                            {
                                testFlag = true;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    else
    {
        if( ((Edge)temp).getStreamType() != null )
            testFlag = true;

    }
    if( isState != ((Edge)temp).isStateStream() )
        testFlag = true;

    if(initialVal != null)
    {
        if( !initialVal.equals( ((Edge)temp).getInitialValue() ) )
            testFlag = true;
    }
    else
    {
        if( ((Edge)temp).getInitialValue() != null )
            testFlag = true;
    }

    if(lantency != null)
    {
        if( !lantency.equals(((Edge)temp).getMet() ) )
            testFlag = true;
    }
    else
    {
        if( ((Edge)temp).getMet() != null )
            testFlag = true;
    }

    }

    if(testFlag)
    {
        localVector.add(temp);
    }
    /*      if( !streamType.equals(((Edge)temp).getStreamType())
        | isState != ((Edge)temp).isStateStream()
        | !initialVal.equals( ((Edge)temp).getInitialValue() )
        | !lantency.equals(((Edge)temp).getMet() )
        )
        {
            localVector.add(temp);
        }
    */
    }
}

}

//inconsistence exits
if(localVector.size()>1)
{
    new StreamCheckTable(localVector);
}

}

} //end of class
package caps.Parser;
import caps.Psdl.*;
import java.util.Enumeration;
import javax.swing.JOptionPane;
import java.io.StringWriter;
import javax.swing.tree.DefaultMutableTreeNode;
/**
 * check PSDL structure restriction before saving it to file.
 * 1.if DFC is Vertex ,check its sub-level Edges for consistence.
 * 2.if DFC is Edge ,check its parent Vertex for consistence.
 * @author Shen-Yi Tao
 * @version 1.0
 */

public class StructureCheck
{
    private static StringBuffer inconsistentRecord = new StringBuffer();

    public StructureCheck()

```

```

{
}
//6/7/00 SYT
/**
 * check the consistence of the PSDL structure from high level(parent node)
 * to lower level(child nodes)
 */
public static void psdlStructureCheck(Vertex root)
{
    for(Enumeration a = root.breadthFirstEnumeration(); a.hasMoreElements();)
    {
        DataFlowComponent DFC =(DataFlowComponent) a.nextElement();
        //if DFC is Vertex ,check its children node for consistence.
        if(DFC instanceof Vertex & !DFC.isLeaf())
        {
            //if the in-edge of vertex is not empty,
            //search inEdge of it's sub-level vertex to find boolean status
            //that "the same name exiting in the lower level".
            if( !((Vertex)DFC).getInEdgesVector().isEmpty())
            {
                checkSubVertexInEdge(((Vertex)DFC) );
            }
            if( !((Vertex)DFC).getOutEdgesVector().isEmpty())
            {
                checkSubVertexOutEdge(((Vertex)DFC) );
            }
        }
        //if DFC is Edge ,it is not root ,and its parent is not root
        //,check its parent node for consistence.
        if( (DFC instanceof Edge) && !((
(DefaultMutableTreeNode)(DFC.getParent()).isRoot()) )
        {
            if( ((Edge)DFC).getSource() instanceof External )
            {
                checkParentVertexInEdge((Edge)DFC);
            }
            if( ((Edge)DFC).getDestination() instanceof External )
            {
                checkParentVertexOutEdge((Edge)DFC);
            }
        }
    }
    if(! (inconsistentRecord.length()==0 ) )
    {
        showErrorDialog(inconsistentRecord.toString());
    }
}
//6/7/00 SYT
/**
 * search inEdge of the parent Vertex's sub-level vertex to find
 * boolean status that "the same name exiting in the lower level".
 * @ parentVertex- Vertex to search
 */
public static void checkSubVertexInEdge(Vertex parentVertex)
{
    String name = "";
    String inconsistentEdgeName = "";

    StringBuffer tempRecord = new StringBuffer();
    boolean inEdgeFound = false;
    for(Enumeration b = parentVertex.getInEdgesVector().elements()
        ; b.hasMoreElements();)
    {
        inEdgeFound = false;
        name = ((DataFlowComponent) b.nextElement()).getLabel();

        for(Enumeration c = parentVertex.children() ; c.hasMoreElements();)
        {
            DataFlowComponent cDFC = (DataFlowComponent) c.nextElement();
            if(cDFC instanceof Edge)
            {
                if(cDFC.getLabel().equals(name) )
                {
                    if( ((Edge)cDFC).getSource() instanceof External
                        | ((Edge)cDFC).getSource()
                            .equals( ((Edge)cDFC).getDestination() )
                    )
                }
            }
        }
    }
}

```

```

        {
            inEdgeFound = true;
        }
    }
}
if(!inEdgeFound)
{
    if(tempRecord.length() == 0)
        tempRecord.append(name);
    else
        tempRecord.append(", " + name);
}
}

if( tempRecord.length() != 0 )
{
    if(inconsistentRecord.length() == 0)
    {
        inconsistentRecord.append("Structure inconsistency was found!\n") ;
    }

    inconsistentRecord.append("Vertex: " + parentVertex.getLabel()
        + " loses some edges in its sub-level as listed.\n "
        + "EXTERNAL -> VERTEX : " + tempRecord.toString() + "\n" );
}
}

//6/7/00 SYT
/**
 * search outEdge of it's sub-level vertex to find boolean status
 * that "the same name existing in the lower level".
 * @ parentVertex- Vertex to search
 */
public static void checkSubVertexOutEdge(Vertex parentVertex)
{
    String name = "";
    String inconsistentEdgeName = "";

    StringBuffer tempRecord = new StringBuffer();
    boolean outEdgeFound=false;
    for(Enumeration b = parentVertex.getOutEdgesVector().elements()
        ; b.hasMoreElements(); )
    {
        outEdgeFound = false;
        name = ((DataFlowComponent) b.nextElement()).getLabel();
        for(Enumeration c = parentVertex.children() ; c.hasMoreElements(); )
        {
            DataFlowComponent cDFC = (DataFlowComponent) c.nextElement();
            if(cDFC instanceof Edge)
            {
                if(cDFC.getLabel().equals(name) )
                {
                    if( ((Edge)cDFC ).getDestination() instanceof External
                        | ((Edge)cDFC ).getSource()
                            .equals( ((Edge)cDFC).getDestination() )
                    )
                    {
                        outEdgeFound = true;
                    }
                }
            }
        }
    }
    if(!outEdgeFound)
    {
        if(tempRecord.length()==0)
            tempRecord.append(name);
        else
            tempRecord.append(", "+name);
    }
}

if( tempRecord.length() != 0 )
{
    if(inconsistentRecord.length() == 0)
    {

```

```

        inconsistentRecord.append("Structure inconsistency was found!\n") ;
    }

    inconsistentRecord.append("Vertex: " + parentVertex.getLabel()
        + " loses some edges in its sub-level as listed.\n "
        + "VERTEX -> EXTERNAL : " + tempRecord.toString() + "\n" );
    }
}

public static void showErrorDialog (String str)
{
    JOptionPane.showMessageDialog (null, str, "Error Message"
        , JOptionPane.ERROR_MESSAGE);

    //clearn all 6/10/00 SYT
    inconsistentRecord.setLength(0);
}

public static void checkParentVertexInEdge(Edge targetEdge)
{
    boolean inEdgeFound = false;
    String name = targetEdge.getLabel();

    if( targetEdge.getParent()!=null)
    {
        for(Enumeration e =
            ((Vertex)targetEdge.getParent()).getInEdgesVector().elements()
            ;e.hasMoreElements();)
        {
            Edge edge = (Edge)(e.nextElement());

            if(edge.getLabel().equals(name) )
                inEdgeFound = true;
        }
        if(!inEdgeFound)
        {
            if(inconsistentRecord.length() == 0)
            {
                inconsistentRecord.append("Structure inconsistency was found!\n") ;
            }

            inconsistentRecord.append("Vertex: " + ((Vertex)targetEdge.getParent())
                .getLabel() + " loses the in-edge as listed.\n "
                + "EXTERNAL -> VERTEX : " + name + "\n" );
        }
    }
}

public static void checkParentVertexOutEdge(Edge targetEdge)
{
    boolean outEdgeFound = false;
    String name = targetEdge.getLabel();

    if( targetEdge.getParent()!=null)
    {
        for(Enumeration e =
            ((Vertex)targetEdge.getParent()).getOutEdgesVector().elements()
            ;e.hasMoreElements();)
        {
            Edge edge = (Edge)(e.nextElement());

            if(edge.getLabel().equals(name) )
                outEdgeFound = true;
        }
        if(!outEdgeFound)
        {
            if(inconsistentRecord.length() == 0)
            {
                inconsistentRecord.append("Structure inconsistency was found!\n") ;
            }

            inconsistentRecord.append("Vertex: " + ((Vertex)targetEdge.getParent())
                .getLabel() + " loses the out-edge as listed.\n "
                + "VERTEX -> EXTERNAL : " + name + "\n" );
        }
    }
}
}
}

```

```

package caps.Psdl;

import javax.swing.tree.DefaultMutableTreeNode;
import java.awt.*;
import caps.GraphEditor.FontConstants;

/**
 * DataFlowComponent is the abstract base class of the Vertex and
 * Edge classes.
 * It extends DefaultMutableTreeNode, so every object of this class is
 * actually a tree node.
 * @author Shen-Yi Tao
 * @version 1.1
 */
public abstract class DataFlowComponent extends DefaultMutableTreeNode
{

    /**
     * The label to display on the DrawPanel
     */
    protected String label;

    /**
     * The unique id of components.
     */
    protected static int UNIQUE_ID = 0;

    /**
     * The id of this component
     */
    protected int id; // Op_num or edge id

    /**
     * The font parameter of the label.
     */
    protected int labelFont;

    /**
     * The font representation of the label.
     */
    protected Font lFont;

    /**
     * The x-offset of the label from the center of the component
     */
    protected int labelXOffset;

    /**
     * The y-offset of the label from the center of the component
     */
    protected int labelYOffset;

    /**
     * The met of a Vertex or the latency of a Stream.
     */
    protected PSDLTime met;

    /**
     * The font parameter of the met label of this component.
     */
    protected int metFont;

    /**
     * The x-offset of the met label from the center of this component.
     */
    protected int metXOffset;

    /**
     * The y-offset of the met label from the center of this component.
     */
    protected int metYOffset;

    /**
     * The font representation of the met (or latency).
     */
    protected Font metlFont; // The real font to display on the screen

```

```

/**
 * The parent of this component.
 */
//protected Vertex parent;

/**
 * The constructor for this class.
 *
 * @param v          The parent vertex of this component
 */
protected DataFlowComponent (Vertex v)
{
    super ();

    labelXOffset = 0;
    labelYOffset = 0;

    metXOffset = 0;
    metYOffset = -40;

    labelFont = 4;
    metFont = 4;
    lFont = new Font ("Courier", Font.PLAIN, 12);
    metlFont = new Font ("Courier", Font.PLAIN, 12);

    setAllowsChildren (false);

    // I don't need parent any more SYT
    //parent = v;                // Sets the parent Vertex
    if (v != null)
    {
        // If not the root operator
        // allow add childs to this tree. SYT 12/29/99
        v.setAllowsChildren (true);
        // add this MutableTreeNode to this tree. SYT 12/29/99
        v.add (this);           // Calls DefaultMutableTreeNode's add method
    }
}

//public Vertex getParentVertex ()
//{
//    return parent;
// }

//add 2/5/00 SYT
public void delete()
{
}

/**
 * Returns the id of this component.
 */
public int getId ()
{
    return id;
}

/**
 * Sets the id of this component to the specified value.
 */
public void setId (int i)
{
    id = i;
}

/**
 * Sets the label of this component to the specified value.
 */
public void setLabel (String s)
{
    label = s;
}

/**
 * Returns the label of this component.
 */
public String getLabel ()
{

```

```

        return label;
    }

    /**
     * Returns the x-component of the offset of the label.
     */
    public int getLabelXOffset ()
    {
        return labelXOffset;
    }

    /**
     * Sets the x-component of the offset of the label to the specified value.
     */
    public void setLabelXOffset (int xLoc)
    {
        labelXOffset = xLoc;
    }

    /**
     * Sets the y-component of the offset of the label to the specified value.
     */
    public void setLabelYOffset (int yLoc)
    {
        labelYOffset = yLoc;
    }

    /**
     * Returns the y-component of the offset of the label.
     */
    public int getLabelYOffset ()
    {
        return labelYOffset;
    }

    /**
     * Changes the label offset to the specified x and y values.
     */
    public void setLabelOffset (int xOffset, int yOffset)
    {
        labelXOffset = labelXOffset + xOffset;
        labelYOffset = labelYOffset + yOffset;
    }

    /**
     * Returns font of the label.
     */
    public Font getlFont ()
    {
        return lFont;
    }

    /**
     * Sets the met (or latency) of this component to the specified value.
     */
    public void setMet (PSDLTime s)
    {
        met = s;
    }

    /**
     * Returns the met (or latency) of this component.
     */
    public PSDLTime getMet ()
    {
        return met;
    }

    /**
     * Returns the x-component of the offset of the met (or latency).
     */
    public int getMetXOffset ()
    {
        return metXOffset;
    }

    /**

```



```

    * Sets the x-component of the offset of the met (or latency)
    * to the specified value.
    */
public void setMetXOffset (int xLoc)
{
    metXOffset = xLoc;
}

/**
 * Sets the y-component of the offset of the met (or latency) to
 * the specified value.
 */
public void setMetYOffset (int yLoc)
{
    metYOffset = yLoc;
}

/**
 * Returns the y-component of the offset of the met (or latency).
 */
public int getMetYOffset ()
{
    return metYOffset;
}

/**
 * Changes the met (or latency) offset to the specified x and y values.
 */
public void setMetOffset (int xOffset, int yOffset)
{
    metXOffset = metXOffset + xOffset;
    metYOffset = metYOffset + yOffset;
}

/**
 * Returns font of the met (or latency).
 */
public Font getMetlFont ()
{
    return metlFont;
}

/**
 * This abstract method is implemented in the subclasses.
 */
public abstract int getX ();

/**
 * This abstract method is implemented in the subclasses.
 */
public abstract int getY ();

/**
 * This abstract method is implemented in the subclasses.
 */
public abstract void moveTo (int xOffset, int yOffset);

/**
 * Returns the name (label) of this component.
 */
public String toString ()
{
    return label;
}

/**
 * Changes the label font index to the specified value.
 */
public void setLabelFontIndex (int f)
{
    labelFont = f;
    int type = ((FontConstants.FONT_VALUES [(f - 1) * 3 + 1].equals("Plain"))
        ? Font.PLAIN : Font.BOLD);
    lFont = new Font (FontConstants.FONT_VALUES [(f - 1) * 3]
        , type , Integer.parseInt (FontConstants.FONT_VALUES [(f - 1) * 3 + 2]));
}

```

```

/**
 * Returns the label font index of this component.
 */
public int getLabelFontIndex ()
{
    return labelFont;
}

/**
 * Changes the met (or latency) font index to the specified value.
 */
public void setMetFontIndex (int f)
{
    metFont = f;
    int type = ((FontConstants.FONT_VALUES [(f - 1) * 3 + 1].equals("Plain"))
        ? Font.PLAIN : Font.BOLD);
    metlFont = new Font (FontConstants.FONT_VALUES [(f - 1) * 3]
        ,type
        ,Integer.parseInt(FontConstants.FONT_VALUES [(f - 1) * 3 + 2]));
}

/**
 * Returns the met (or latency) font index of this component.
 */
public int getMetFontIndex ()
{
    return metFont;
}

} // End of class DataFlowComponent
package caps.Psdl;

import java.util.Vector;
import java.util.Enumeration;
import java.io.*;

public class DataTypes
{
    private Vector types;

    private Vector specs;

    private Vector impls;

    public DataTypes ()
    {
        types = new Vector ();
        specs = new Vector ();
        impls = new Vector ();
    }

    // This will be called from the builder
    public void addType (String name, String spec, String impl)
    {
        if (!exists (name))
        {
            types.addElement (name);
            specs.addElement (spec);
            impls.addElement (impl);
        }
    }

    // This will be called when a new edge is created
    public void addType (String name)
    {
        if (!exists (name) && !isPredefined (name))
        {
            types.addElement (name);
            specs.addElement ("\nEND");
            impls.addElement ("ada " + name + "\nEND");
        }
    }

    public boolean exists (String name)
    {
        boolean flag = false;

```

```

        for (Enumeration enum = types.elements (); enum.hasMoreElements ();)
        {
            if (name.equals((String) enum.nextElement ()))
                flag = true;
        }
        return flag;
    }

    public boolean isPredefined (String str)
    {
        if (str.equalsIgnoreCase ("boolean") || str.equalsIgnoreCase ("character")
            || str.equalsIgnoreCase ("string") || str.equalsIgnoreCase ("integer")
            || str.equalsIgnoreCase ("real") || str.equalsIgnoreCase ("exception"))
            return true;
        else
            return false;
    }

    // this is called when reafing form the file for the first time
    public void buildTypes (File file)
    {
        StreamTokenizer tok = null;
        try
        {
            tok = new StreamTokenizer (new FileReader (file));
        }
        catch (FileNotFoundException ex)
        {
            System.out.println (ex);
        }
        build (tok);
    }

    // called when building types in the editor
    public void buildTypes (String s)
    {
        StreamTokenizer tok = null;
        tok = new StreamTokenizer (new StringReader (s));
        build (tok);
    }

    private void build (StreamTokenizer tok)
    {
        removeElements ();
        String str;
        String tempStr;
        tok.wordChars (33, 126);
        tok.eolIsSignificant (true);
        int tokType;
        int counter = 0;

        try
        {
            while ((tokType = tok.nextToken ()) != StreamTokenizer.TT_EOF)
            {
                if (tokType == StreamTokenizer.TT_WORD
                    && tok.sval.equalsIgnoreCase ("TYPE"))
                {
                    tempStr = getNextToken (tok);
                    str = tempStr;
                    types.addElement (str);
                    do
                    {
                        tempStr = getNextToken (tok);
                    } while (!tempStr.equalsIgnoreCase ("SPECIFICATION"));
                    str = str.concat (tempStr);
                    counter++;
                    do
                    {
                        tempStr = getNextToken (tok);
                        if (tempStr.equalsIgnoreCase ("SPECIFICATION"))
                            counter++;
                        else if (tempStr.equalsIgnoreCase ("END"))
                            counter--;
                        str = str.concat (tempStr);
                        if (tempStr != "\n")
                            str = str.concat (" ");
                    }
                }
            }
        }
    }

```

```

        } while (counter > 0 || !tempStr.equalsIgnoreCase ("END"));
        specs.addElement (str);
        do
        {
            tempStr = getNextToken (tok);
        } while (!tempStr.equalsIgnoreCase ("IMPLEMENTATION"));
        str = "";
        counter++;
        do
        {
            tempStr = getNextToken (tok);
            if (tempStr.equalsIgnoreCase ("IMPLEMENTATION"))
                counter++;
            else if (tempStr.equalsIgnoreCase ("END"))
                counter--;
            str = str.concat (tempStr);
            if (tempStr != "\n")
                str = str.concat (" ");
        } while (counter!= 0 || !tempStr.equalsIgnoreCase ("END"));
        impls.addElement (str);
    }
}
}
catch (IOException ex)
{
    System.out.println (ex);
}
}

public String getNextToken (StreamTokenizer tok) throws IOException
{
    String str = "";
    tok.nextToken ();
    if (tok.ttype == StreamTokenizer.TT_EOL)
        str = "\n";
    else if (tok.ttype == StreamTokenizer.TT_WORD)
        str = str.concat (tok.sval);
    return str;
}

public void removeElements ()
{
    types.removeAllElements ();
    specs.removeAllElements ();
    impls.removeAllElements ();
}

public String toString ()
{
    String str = "";
    int numberOfTypes = types.size ();
    for (int ix = 0; ix < numberOfTypes; ix++)
    {
        str = str.concat ("TYPE " + (String) types.elementAt (ix) + "\n");
        str = str.concat ("SPECIFICATION "
            + (String) specs.elementAt (ix) + "\n");
        str = str.concat ("IMPLEMENTATION "
            + (String) impls.elementAt (ix) + "\n\n");
    }
    return str;
}

} // End of the class DataTypes.
package caps.Psdl;

import java.util.Vector;
import java.util.Enumeration;
import java.awt.Point;
import java.io.StringReader;
import java.io.StreamTokenizer;
import java.io.IOException;

/**
 * Edge represents a stream in the data flow diagram
 * It is also a TreeNode object
 */

```

```

* @author Shen-Yi Tao
* @version 1.1
*/

public class Edge extends DataFlowComponent
{
    //add SYT
    /**
     * unique ID used to identify this Edge instead of
     * id which may be modified
     */
    private static int EdgeID;

    //add SYT
    /**
     * used by clone . (solve static address problem)
     * same value as EdgeID
     * this will be used when undo and redo
     */
    private int CloneEdgeID;

    /**
     * The source Vertex of this stream.
     */
    protected Vertex source;

    /**
     * The destination Vertex of this stream.
     */
    protected Vertex destination;

    /**
     * The vector that holds the control points of this stream.
     */
    protected Vector Points;

    /**
     * The type name of the stream.
     */
    protected String streamType;

    /**
     * The initial value of the stream.
     */
    protected String initialValue;

    /**
     * True if this is a state stream.
     */
    protected boolean isState;

    /**
     * The x location of this stream in the DrawPanel.
     */
    protected int x;

    /**
     * The y location of this stream in the DrawPanel.
     */
    protected int y;

    /**
     * The index of the handle that the mouse is pressed on.
     */
    protected int selectedHandleIndex;

    /**
     * The constructor for this class.
     *
     * @param v the parent vertex of this edge.
     */
    public Edge (int xLocation, int yLocation, Vertex v)
    {
        super (v);
        source = null;
        destination = null;
    }

```

```

        Points = new Vector (0, 2);

        Points.addElement (new Point (xLocation, yLocation));
        streamType = "undefined_type";
        initialValue = "";
        setLabel ("unnamed_stream_" + UNIQUE_ID++);
        EdgeID++;
        CloneEdgeID = EdgeID;

        id = ++UNIQUE_ID;

        isState = false;
        met = null;
        setX (xLocation);
        setY (yLocation);
    }
    //1/21/00 SYT
    /**
     * clone a Points vector (slove share address problem)
     * and change address of this Points vector
     */
    public void clonePoints()
    {
        Vector V =new Vector(0,2);
        for(Enumeration e = Points.elements(); e.hasMoreElements();)
        {
            Point P= (Point)((Point)(e.nextElement()).clone());
            V.add(P);
        }
        Points=V;
    }

    //add 1/21/00 SYT
    /**
     * get current EdgeID
     */
    public int getEdgeID()
    {
        return EdgeID;
    }

    //add 1/21/00 SYT
    /**
     * get colone edgeID (used in undo & redo)
     */
    public int getCloneEdgeID()
    {
        return CloneEdgeID;
    }

    /**
     * Relocates the stream when the stream is moved with other objects.
     */
    public void moveTo (int xOffset, int yOffset)
    {
        Point p;
        for (Enumeration enum = Points.elements (); enum.hasMoreElements ());
        {
            p = (Point) enum.nextElement ();
            p.x = p.x + xOffset;
            p.y = p.y + yOffset;
        }
        correctLabelOffset ();
    }

    /**
     * called when one of the handles of the stream is dragged
     * in the DrawPanel.
     */
    public void reShape (int xLocation, int yLocation)
    {
        if (selectedHandleIndex == 0) // the source
            return;

        if (Points.size () == 3)
        {
            // has only one control point, add more
            Point begin = (Point) Points.elementAt (0);

```

```

        Point end = (Point) Points.elementAt (2);
        int xDiff = (end.x - begin.x) / 6;
        int yDiff = (end.y - begin.y) / 6;
        Points.removeElementAt (1);
        for (int index = 1; index < 6; index++)
        {
            Points.add (index, new Point (begin.x + xDiff * index, begin.y
            + yDiff * index));
        }
        selectedHandleIndex = 3;
    }

    Point p = (Point) Points.elementAt (selectedHandleIndex);
    Point prev = (Point) Points.elementAt (selectedHandleIndex - 1);
    Point next = (Point) Points.elementAt (selectedHandleIndex + 1);
    Point middle;

    int diffX = xLocation - p.x;
    int diffY = yLocation - p.y;
    p.x = p.x + diffX;
    p.y = p.y + diffY;

    if (selectedHandleIndex == 1)
    {
        next.x = next.x + diffX * 2;
        next.y = next.y + diffY * 2;

        Point nextControl= (Point) Points.elementAt (selectedHandleIndex + 3);
        middle = (Point) Points.elementAt (selectedHandleIndex + 2);
        middle.x = (next.x + nextControl.x) / 2;
        middle.y = (next.y + nextControl.y) / 2;
    }
    else if (selectedHandleIndex == Points.size () - 2)
    {
        prev.x = prev.x + diffX * 2;
        prev.y = prev.y + diffY * 2;

        Point prevControl= (Point) Points.elementAt (selectedHandleIndex - 3);
        middle = (Point) Points.elementAt (selectedHandleIndex - 2);
        middle.x = (prev.x + prevControl.x) / 2;
        middle.y = (prev.y + prevControl.y) / 2;
    }
    else
    {
        prev.x = prev.x + diffX;
        prev.y = prev.y + diffY;
        Point prevControl= (Point) Points.elementAt (selectedHandleIndex - 3);
        middle = (Point) Points.elementAt (selectedHandleIndex - 2);
        middle.x = (prev.x + prevControl.x) / 2;
        middle.y = (prev.y + prevControl.y) / 2;

        next.x = next.x + diffX;
        next.y = next.y + diffY;
        Point nextControl= (Point) Points.elementAt (selectedHandleIndex + 3);
        middle = (Point) Points.elementAt (selectedHandleIndex + 2);
        middle.x = (next.x + nextControl.x) / 2;
        middle.y = (next.y + nextControl.y) / 2;
    }
    correctLabelOffset ();
}

/**
 * Changes the x value of the stream to the specified value.
 */
public void setX (int newX)
{
    x = newX;
}

/**
 * Changes the y value of the stream to the specified value.
 */
public void setY (int newY)
{
    y = newY;
}

```

```

/**
 * Changes selectedHandleIndex to the specified value.
 */
public void setSelectedHandleIndex (int i)
{
    selectedHandleIndex = i;
}

/**
 * Returns the x value of this stream.
 */
public int getX ()
{
    return x;    // **** Pending ****
}

/**
 * Returns the y value of this stream.
 */
public int getY ()
{
    return y;    // **** Pending ****
}

/**
 * Returns the source Vertex of this stream.
 */
public Vertex getSource ()
{
    return source;
}

/**
 * Sets the source Vertex of this stream to the specified value.
 */
public void setSource (Vertex v)
{
    source = v;
}

/**
 * Returns the destination Vertex of this stream.
 */
public Vertex getDestination ()
{
    return destination;
}

/**
 * Sets the destination Vertex of this stream to the specified value.
 */
public void setDestination (Vertex v)
{
    destination = v;
}

/**
 * Returns the type of this stream.
 */
public String getStreamType ()
{
    return streamType;
}

/**
 * Sets the type of this stream to the specified value.
 */
public void setStreamType (String type)
{
    streamType = type;
}

/**
 * Returns true if this is a state stream.
 */
public boolean isStateStream ()
{

```



```

        return isState;
    }

    /**
     * Changes the isState field to the specified value.
     */
    public void setStateStream (boolean flag)
    {
        isState = flag;
    }

    /**
     * Returns the initial value of this stream.
     */
    public String getInitialValue ()
    {
        return initialValue;
    }

    /**
     * Sets the initial value of this stream to the specified value.
     */
    public void setInitialValue (String str)
    {
        initialValue = str;
    }

    /**
     * Adds a new point to the control Points.
     * Also adds the middle point of the control Points.
     *
     * @param x the x component of the new control point.
     * @param y the y component of the new control point.
     */
    public void addPoint (int x, int y)
    {
        Point p = (Point) Points.lastElement (); // the last element
        Point middle = new Point ((x + p.x) / 2, (y + p.y) / 2);
        Points.addElement (middle);
        Points.addElement (new Point (x, y));
    }

    /**
     * Returns the control Points vector.
     */
    public Vector getPoints ()
    {
        return Points;
    }

    /**
     * Sets the location of this stream to the middle control point.
     */
    // Pending this is called by so many methods needlessly
    public void correctLabelOffset ()
    {
        Point p = (Point) Points.elementAt (Points.size () / 2); // The middle point
        in the vector
        setX (p.x + 10); setY (p.y - 10);
    }

    // SYT
    /**
     * modified for debugging
     * Locates the ending Points of this stream on
     * the perimeter of the source and destination.
     */
    public void correctEndingPoints ()
    {
        //following code cause ArrayIndexOutOfBoundsException 8/14/00 modified SYT
        if (Points.size() >= 2)
        {
            Point p1 = source.getIntersectionPoint ((Point) Points.elementAt (1));
            Point p2 = destination.getIntersectionPoint ((Point) Points.elementAt
(Points.size () - 2));
            Point p3;
            Point middle;

```

```

        Points.setElementAt (p1, 0);
        p3 = (Point) Points.elementAt (2);
        middle = (Point) Points.elementAt (1);
        middle.setLocation ((p1.x + p3.x) / 2, (p1.y + p3.y) / 2);

        Points.setElementAt (p2, Points.size () - 1);
        p3 = (Point) Points.elementAt (Points.size () - 3);
        middle = (Point) Points.elementAt (Points.size () - 2);
        middle.setLocation ((p2.x + p3.x) / 2, (p2.y + p3.y) / 2);

        correctLabelOffset ();
    }

}

/**
 * Called to extract a string representation of the control Points.
 * Constructs the Points vector from the string expression.
 */
// called to build the Points vector
public void setInitialControlPoints (String exp)
{
    Points.removeAllElements ();

    exp = exp.substring (1, exp.length () - 1);
    exp.trim ();
    StringReader reader = new StringReader (exp);
    StreamTokenizer tok = new StreamTokenizer (reader);
    int tokType;

    if (source instanceof External)
    {
        try
        {
            tok.nextToken ();
            source.setX ((int) tok.nval);
            tok.nextToken ();
            source.setY ((int) tok.nval);
        }
        catch (IOException ex)
        {
            System.out.println (ex);
        }
    }

    Points.addElement (new Point (source.getX (), source.getY ()));

    try
    {
        while ((tokType = tok.nextToken ()) != StreamTokenizer.TT_EOF)
        {
            int x = (int) tok.nval;
            tok.nextToken ();
            int y = (int) tok.nval;
            addPoint (x, y);
        }
    }
    catch (IOException ex)
    {
        System.out.println (ex);
    }

    if (destination instanceof External)
    {
        destination.setX (((Point) Points.lastElement ().x);
        destination.setY (((Point) Points.lastElement ().y);
    }
    else
    {
        addPoint (destination.getX (), destination.getY ());
        correctEndingPoints ();
    }
}

/**
 * Deletes this stream.
 */
public void delete (boolean deletingInEdge)

```

```

    {
        if (deletingInEdge)
        {
            source.removeOutEdge (this);
        }
        else
        {
            destination.removeInEdge (this);
        }
        deleteHelper ();
    }

    /**
     * Deletes this stream.
     */
    public void delete ()
    {
        source.removeOutEdge (this);
        destination.removeInEdge (this);
        deleteHelper ();
    }

    /**
     * Helper method to delete the stream.
     */
    public void deleteHelper ()
    {
        //close 7/28/00 SYT
        //error : ArrayIndexOutOfBounds and state statement is incomplete.
        //states statement generated by logic in createPsdl.java
        // I do not need elementAt(2) any more
        /*
        if (isStateStream ())
        {
            int index = -1;
            label :
            for (Enumeration enum = parent.children (); enum.hasMoreElements ();)
            { // trying to find index
                DataFlowComponent dfc = (DataFlowComponent) enum.nextElement ();
                if (dfc instanceof Edge && ((Edge) dfc).isStateStream ())
                    index++;
                if (dfc.equals (this))
                    break label;
            }

            ((Vector) ((Vertex) parent).getSpecReqmts ().elementAt (2))
                .removeElementAt (index);
        } */
        if (source instanceof External)
            source.delete ();
        if (destination instanceof External)
            destination.delete ();
        removeFromParent ();
    }

} // End of the class Edge.
package caps.Psdl;

import java.awt.Point;
/**
 * Represent External of PSDL
 * @author Shen-Yi Tao
 * @version 1.1
 */
public class External extends Vertex
{
    public External (int xLocation, int yLocation, Vertex v)
    {
        super (xLocation, yLocation, v, false);

        met = null;
        setLabel ("EXTERNAL");

        labelYOffset = 10;

        x = xLocation;

```

```

        y = yLocation;
        width = 0;
        height = 0;

        removeFromParent ();
    }

    public Point getIntersectionPoint (Point p)
    {
        return new Point (x, y);
    }

} // End of the class External
package caps.Psdl;

/**
 * This class represents a combination of time value from an
 * integer that represents
 * the time and another integer that represents the unit.
 *
 * @author Shing_Yi Tao
 * @version 1.1
 */
public class PSDLTime extends Object
{
    /**
     * The constant value for microseconds.
     */
    public final static int microsec = 0;

    /**
     * The constant value for milliseconds.
     */
    public final static int ms = 1;

    /**
     * The constant value for seconds.
     */
    public final static int sec = 2;

    /**
     * The constant value for minutes.
     */
    public final static int min = 3;

    /**
     * The constant value for hours.
     */
    public final static int hours = 4;

    /**
     * The value of the time.
     */
    private int value;

    /**
     * The units of the time.
     */
    private int units;

    /**
     * The constructor for this class.
     */
    public PSDLTime()
    {
        value = 0;
        units = ms;
    }

    /**
     * The constructor for this class.
     *
     * @param timeValue the value of the time.
     * @param timeUnits the unit of the time.
     */
    public PSDLTime(int timeValue, int timeUnits)

```

```

    {
        value = timeValue;
        units = timeUnits;
    }

    /**
     * return time in second   SYT
     */
    public double getTimeInSeconds()
    {
        double value = 0;
        switch(getTimeUnits())
        {
            case 0:
                value = getTimeValue() * 0.000001;
                break;
            case 1:
                value = getTimeValue() * 0.001;
                break;
            case 2:
                value = getTimeValue() * 1;
                break;
            case 3:
                value = getTimeValue() * 60;
                break;
            case 4:
                value = getTimeValue() * 3600;
                break;
        }
        return value;
    }

    /**
     * Returns the time time value of this object.
     */
    public int getTimeValue()
    {
        return value;
    }

    /**
     * Sets the time value to the specified argument.
     */
    public void setTimeValue(int timeValue)
    {
        value = timeValue;
    }

    /**
     * Returns the time units of this object.
     */
    public int getTimeUnits()
    {
        return units;
    }

    /**
     * Sets the time unit to the specified argument.
     */
    public void setTimeUnits(int timeUnits)
    {
        units = timeUnits;
    }

    /**
     * bugs :
     * 8/12/00 modified SYT
     * Sets the time unit to the specified argument.
     */
    public void setTimeUnits (String u)
    {
        //System.out.println("in:"+u);
        //if (u == "microsec")
        if (u.equalsIgnoreCase("microsec"))
            units = microsec;
        //else if (u == "ms")
        else if (u.equalsIgnoreCase("ms"))

```

```

        units = ms;
    //else if (u == "sec")
    else if (u.equalsIgnoreCase("sec"))
        units = sec;
    //else if (u == "min")
    else if (u.equalsIgnoreCase("min"))
        units = min;
    //else if (u == "hours")
    else if (u.equalsIgnoreCase("hours"))
        units = hours;
    //System.out.println("out:"+Integer.toString(units));
}

/**
 * Returns a string representation of this object.
 *
 * @return the string representation in the form of "12 sec"
 */
public String toString()
{
    String unitString;
    switch (units)
    {
        case 0 : unitString = "microsec";
            break;
        case 1 : unitString = "ms";
            break;
        case 2 : unitString = "sec";
            break;
        case 3 : unitString = "min";
            break;
        case 4 : unitString = "hours";
            break;
        default : unitString = "undefined";
    }
    return String.valueOf(value) + " " + unitString;
}

} // End of the class PSDLTime
/**
 * 11/10/99 SYT
 * modification for a new grammar
 * This class represents a terminator or an operator.
 * It holds the data structures that represent the constructs for the Vertex.
 */
package caps.Psdl;

import java.awt.Font;
import java.util.*;
import java.awt.Point;
import java.io.BufferedReader;
import java.io.StringReader;
import java.io.IOException;
// 11/9/99 SYT
// add
import caps.Parser.GrammarCheck;

public class Vertex extends DataFlowComponent
{
    //add 7/3/00 SYT
    /**
     * critical status: 1-"Hard", 2-"Soft" , 3-"none"
     */
    private int criticalStatus;
    //add 4/1/00 SYT
    /**
     * network label
     */
    private String networkLabel;

    // add by SYT 3/10/00
    /**
     * current display parent vertex.
     */
    private boolean isParent;

```

```

//add SYT
/**
 * unique ID used to identify this Edge instead of id which may be modified
 */
private static int vertexID;

//add SYT
/**
 * used by cloning a Vertex . (solve static sharing address problem)
 * this will be used when undo and redo
 */
private int cloneVertexID;

/**
 * The constant value for the initial radius of a Vertex.
 */
public static final int INITIAL_RADIUS = 35;

/**
 * The constant value for non-time critical Vertices.
 */
public static final int NON_TIME_CRITICAL = 0;

/**
 * The constant value for periodic Vertices.
 */
public static final int PERIODIC = 1;

/**
 * The constant value for sporadic Vertices.
 */
public static final int SPORADIC = 2;

/**
 * The constant value for unprotected Vertices.
 */
public static final int UNPROTECTED = 0;

/**
 * The constant value for Vertices that have "BY SOME" triggering construct.
 */
public static final int BY_SOME = 1;

/**
 * The constant value for Vertices that have "BY ALL" triggering construct.
 */
public static final int BY_ALL = 2;

/**
 * True if this Vertex is a terminator.
 */
protected boolean terminator;

/**
 * The x-location of this component on the DrawPanel
 */
protected int x;

/**
 * The y-location of this component on the DrawPanel
 */
protected int y;

/**
 * The width of this component.
 * It serves as the radius of an operator and the width
 * of a terminator width of operator .cap
 */
protected int width;

/**
 * The height of this component.
 */
protected int height;

/**

```

```

    * The color parameter of this component.
    */
    protected int color;

    protected Vector metReqmts;

    protected PSDLTime period;

    protected Vector periodReqmts;

    protected PSDLTime finishWithin;

    protected Vector finishWithinReqmts;

    protected PSDLTime mcp;

    protected Vector mcpReqmts;

    protected PSDLTime mrt;

    protected Vector mrtReqmts;

    protected int timingType;

    protected int triggerType;

    protected Vector triggerReqmts;

    protected Vector triggerStreamsList;

    protected String ifCondition;

    protected String outputGuardList;

    protected String exceptionGuardList;

    protected String exceptionList;

    protected String timerOpList;

    protected Vector keywordList;

    protected String informalDesc;

    protected String formalDesc;

    protected Vector inEdges;

    protected Vector outEdges;

    protected String impLanguage;

    protected Vector timerList;

    protected String graphDesc;

    protected String genericList;

    protected Vector specReqmts;

    // 11/28/99 SYT
    // this is no longer used.
    // and has been replaced by getID()-1.
    // before delete this must delete releated item on psdlBuilder.jj first.
    protected int idExtension;

    /**
     * The constructor for this class.
     *
     * @param xlocation The x component of the location of this component.
     * @param ylocation The y component of the location of this component.
     * @param v          The parent vertex of this component.
     * @param t          true if this component is a terminator.
     */
    public Vertex (int xLocation, int yLocation, Vertex v, boolean t)
    {

```



```

//call constructor of DataFlowComponent
super (v);
//8/22/00 SYT
//initialized value = parent's value
if(v== null)
criticalStatus =3;
else
criticalStatus = v.getCriticalStatus();

//4/1/00 SYT
//change 7/16/00 SYT
//networkLabel ="";
networkLabel ="local_network";
//initialize "Is parent flag to false." 3/10/00 SYT
isParent = false;
//add 1/22/00 SYT
vertexID++;
//add SYT
cloneVertexID =vertexID;
inEdges = new Vector (0);
outEdges = new Vector (0);

terminator = t;

color = 62; // initially white

//change 7/19/00 SYT
//timingType = NON_TIME_CRITICAL;
if(v == null)
timingType = NON_TIME_CRITICAL;
else
{
    if(v.isRoot())
    {
        timingType = NON_TIME_CRITICAL;
        met = null;
    }
    else
    {
        if(v.getTimingType() == 0)
        {
            timingType = v.getTimingType();
            met = null;
        }
        else
        {
            timingType = v.getTimingType();
            //set MET to 0 8/3/00 SYT
            PSDLTime temp = new PSDLTime();
            setMet(temp);
        }
    }
}
triggerType = UNPROTECTED;

//met = null;

if (v == null)
{
    setLabel ("root_" + UNIQUE_ID++);
}
else if (isTerminator ())
{
    setLabel ("terminator_" + UNIQUE_ID++);
    met = new PSDLTime ();
}

else
{
    setLabel ("operator_" + UNIQUE_ID++);
}

setWidth (INITIAL_RADIUS * 2);

if (getParent () == null) //if this is the root
    id = ++UNIQUE_ID;
else

```

```

    {
        UNIQUE_ID++;          // op_num
        id = ++UNIQUE_ID;
    }
    //debug 8/13/00 SYT
    //period = null;
    //if parent is PERIODIC
    if(v == null)
        period = null;
    else if(v.getTimingType()==1)
        period = v.getPeriod();
    else
        period = null;

    finishWithin = null;
    mcp = null;
    mrt = null;
    metReqmts = new Vector (0, 2);
    periodReqmts = new Vector (0, 2);
    finishWithinReqmts = new Vector (0, 2);
    mcpReqmts = new Vector (0, 2);
    mrtReqmts = new Vector (0, 2);
    triggerReqmts = new Vector (0, 2);
    triggerStreamsList = new Vector (0, 2);
    ifCondition = "";
    outputGuardList = "";
    exceptionGuardList = "";
    exceptionList = "";
    timerOpList = "";
    keywordList = new Vector (0, 2);
    informalDesc = "";
    formalDesc = "";
    timerList = new Vector (0, 2);
    graphDesc = "";
    genericList = "";
    specReqmts = new Vector (0, 2);
    specReqmts.addElement (new Vector (0, 2));
    specReqmts.addElement (new Vector (0, 2));
    specReqmts.addElement (new Vector (0, 2));

    impLanguage = "ada";

    x = xLocation;
    y = yLocation; // Set the location of the component
}

//add 7/3/00 SYT
/**
 * set criticalStatus
 */
public void setCriticalStatus(int b)
{
    criticalStatus = b;
}

//add 7/3/00 SYT
/**
 * get criticalStatus
 */
public int getCriticalStatus()
{
    return criticalStatus;
}

//add 4/1/00 SYT
/**
 * set network label
 */
public void setNetWorkLabel(String str)
{
    networkLabel=str;
}

//add 4/1/00 SYT
/**
 *get network label
 */

```

```

public String getNetWorkLabel()
{
    return networkLabel;
}
//add 3/10/00 SYT
/**
 * setup status of the node
 */
public void setIsParent(boolean b)
{
    isParent=b;
}

//add 3/10/00 SYT
/**
 * is parent node
 */
public boolean getIsParent()
{
    return isParent;
}

//add 1/23/00 SYT
/**
 * get clone vertex id
 */
public int getCloneVertexID()
{
    return cloneVertexID;
}

//add. 1/21/00 SYT
/**
 * clone met reauirement
 */
public void cloneMetReqmts()
{
    Vector v=(Vector)metReqmts.clone();
    metReqmts=v;
}

//add. 1/21/00 SYT
/**
 * clone period requirement
 */
public void clonePeriodReqmts()
{
    Vector v=(Vector)periodReqmts.clone();
    periodReqmts=v;
}

//add 1/22/00 SYT
/**
 * get current vertex id
 */
public int getVertexID()
{
    return vertexID;
}

//add SYT
/**
 * get in-edge vector of this vertex
 */
public Vector getInEdgesVector()
{
    return inEdges;
}

//add SYT
/**
 * get out-edge vector of this vertex
 */
public Vector getOutEdgesVector()
{
    return outEdges;
}

```

```

/**
 * Sets the location of this component on the screen.
 * Also corrects the location of the ending streams.
 *
 * @param xLocation The new x component of the location on the drawpanel
 * @param yLocation The new y component of the location on the drawpanel
 */
public void setLocation (int xOffset, int yOffset)
{
    moveTo (xOffset, yOffset);
    correctInOutputStreams ();
}

/**
 * Sets the location of this component on the screen.
 *
 * @param xLocation The new x component of the location on the drawpanel
 * @param yLocation The new y component of the location on the drawpanel
 */
public void moveTo (int xOffset, int yOffset)
{
    x = x + xOffset;
    y = y + yOffset;
}

/**
 * Returns true if this component is a terminator.
 *
 * @return true if this component is a terminator.
 */
public boolean isTerminator ()
{
    return terminator;
}

/**
 * Sets this component as a terminator or a stream.
 * Also changes the width of the component.
 *
 * @param b.
 */
public void setTerminator (boolean b)
{
    terminator = b;
    setWidth (width);
}

/**
 * Corrects the ending points of the in and out streams of this component.
 */
public void correctInOutputStreams ()
{
    for (Enumeration enum = inEdges.elements (); enum.hasMoreElements ();)
    {
        ((Edge) enum.nextElement ().correctEndingPoints ();
    }
    for (Enumeration enum = outEdges.elements (); enum.hasMoreElements ();)
    {
        ((Edge) enum.nextElement ().correctEndingPoints ();
    }
}

/**
 * Returns the width of this component.
 *
 * @return the width of this component.
 */
public int getWidth ()
{
    return width;
}

/**
 * Changes the width of this component.
 *
 * @param w the new width of this component.

```

```

    */
    public void setWidth (int w)
    {
        width = w;
        if (isTerminator ())
            height = (int) (width / 1.4d);
        else
            height = width;
    }

    /**
     * Returns the height of this component.
     *
     * @return the height of this component.
     */
    public int getHeight ()
    {
        return height;
    }

    /**
     * Returns the x component of the location of this Vertex
     *
     * @return x
     */
    public int getX ()
    {
        return x;
    }

    /**
     * Changes the x component of the location of this Vertex
     *
     * @param xLoc.
     */
    public void setX (int xLoc)
    {
        x = xLoc;
    }

    /**
     * Changes the y component of the location of this Vertex.
     *
     * @param yLoc.
     */
    public void setY (int yLoc)
    {
        y = yLoc;
    }

    /**
     * Returns the y component of the location of this Vertex.
     *
     * @return y
     */
    public int getY ()
    {
        return y;
    }

    /**
     * Changes the color value for this Vertex.
     *
     * @param c the new color value.
     */
    public void setColor (int c)
    {
        color = c;
    }

    /**
     * Returns the color value for this Vertex.
     *
     * @return the color value of the Vertex.
     */
    public int getColor ()
    {

```

```

        return color;
    }
    //add 1/21/00 SYT
    /**
     * clone a new InEdges for undo and redo
     * for changing address
     */
    public void cloneInEdges()
    {
        Vector newInEdges = new Vector(1);
        inEdges=newInEdges;
    }

    //add 1/21/00 SYT
    /**
     * clone a new InEdges for undo and redo
     * for changing address
     */
    public void cloneOutEdges()
    {
        Vector newOutEdges = new Vector(1);
        outEdges=newOutEdges;
    }

    /**
     * Adds a new Edge to the inEdges Vector.
     *
     * @param e the new inEdge.
     */
    public void addInEdge (Edge e)
    {
        inEdges.addElement (e);
        ((Vector) specReqmts.elementAt (0)).addElement ("");
    }

    /**
     * Removes an Edge from the inEdges Vector.
     *
     * @param e the inEdge to be removed.
     */
    public void removeInEdge (Edge e)
    {
        int index = inEdges.indexOf (e);

        inEdges.removeElementAt (index);
        ((Vector) specReqmts.elementAt (0)).removeElementAt (index);
    }

    /**
     * Adds a new Edge to the outEdges Vector.
     *
     * @param e the new outEdge.
     */
    public void addOutEdge (Edge e)
    {
        outEdges.addElement (e);
        ((Vector) specReqmts.elementAt (1)).addElement ("");
    }

    /**
     * Removes an Edge from the outEdges Vector.
     *
     * @param e the outEdge to be removed.
     */
    public void removeOutEdge (Edge e)
    {
        int index = outEdges.indexOf (e);
        outEdges.removeElementAt (index);
        ((Vector) specReqmts.elementAt (1)).removeElementAt (index);
    }

    /**
     * Returns the timing type of this Vertex.
     */
    public int getTimingType ()
    {
        return timingType;
    }

```

```

/**
 * Sets the timing type to the specified value.
 */
public void setTimingType (int type)
{
    timingType = type;
}

/**
 * Returns the triggering type of this Vertex.
 */
public int getTriggerType ()
{
    return triggerType;
}

/**
 * Sets the triggering type to the specified value.
 */
public void setTriggerType (int type)
{
    triggerType = type;
}

/**
 * Returns the period value of this Vertex.
 */
public PSDLTime getPeriod ()
{
    return period;
}

/**
 * Returns the finish within value of this Vertex.
 */
public PSDLTime getFinishWithin ()
{
    return finishWithin;
}

/**
 * Returns the mcp value of this Vertex.
 */
public PSDLTime getMcp ()
{
    return mcp;
}

/**
 * Returns the mrt value of this Vertex.
 */
public PSDLTime getMrt ()
{
    return mrt;
}

/**
 * Sets the period to the specified value.
 */
public void setPeriod (PSDLTime p)
{
    period = p;
}

/**
 * Sets the finish within to the specified value.
 */
public void setFinishWithin (PSDLTime fw)
{
    finishWithin = fw;
}

/**
 * Sets the mcp to the specified value.
 */
public void setMcp (PSDLTime m)
{

```

```

        mcp = m;
    }

    /**
     * Sets the mrt to the specified value.
     */
    public void setMrt (PSDLTime mr)
    {
        mrt = mr;
    }

    /**
     * Returns the implementation language of this Vertex.
     */
    public String getImpLanguage ()
    {
        return impLanguage;
    }

    /**
     * Sets the implementation language to the specified value.
     */
    public void setImpLanguage (String s)
    {
        impLanguage = s;
    }

    /**
     * Returns the met requirements of this Vertex.
     */
    public Vector getMetReqmts ()
    {
        return metReqmts;
    }

    /**
     * Sets the met requirements to the specified value.
     */
    public void setMetReqmts (Vector v)
    {
        metReqmts = v;
    }

    /**
     * Returns the period requirements of this Vertex.
     */
    public Vector getPeriodReqmts ()
    {
        return periodReqmts;
    }

    /**
     * Sets the period requirements to the specified value.
     */
    public void setPeriodReqmts (Vector v)
    {
        periodReqmts = v;
    }

    /**
     * Returns the finish within requirements of this Vertex.
     */
    public Vector getFinishWithinReqmts ()
    {
        return finishWithinReqmts;
    }

    /**
     * Sets the finish within requirements to the specified value.
     */
    public void setFinishWithinReqmts (Vector v)
    {
        finishWithinReqmts = v;
    }

    /**
     * Returns the mcp requirements of this Vertex.

```



```

    */
    public Vector getMcpReqmts ()
    {
        return mcpReqmts;
    }

    /**
     * Sets the mcp requirements to the specified value.
     */
    public void setMcpReqmts (Vector v)
    {
        mcpReqmts = v;
    }

    /**
     * Returns the mrt requirements of this Vertex.
     */
    public Vector getMrtReqmts ()
    {
        return mrtReqmts;
    }

    /**
     * Sets the mrt requirements to the specified value.
     */
    public void setMrtReqmts (Vector v)
    {
        mrtReqmts = v;
    }

    /**
     * Returns the trigger requirements of this Vertex.
     */
    public Vector getTriggerReqmts ()
    {
        return triggerReqmts;
    }

    /**
     * Sets the trigger requirements to the specified value.
     */
    public void setTriggerReqmts (Vector v)
    {
        triggerReqmts = v;
    }

    /**
     * Returns the triggering streams of this Vertex.
     */
    public Vector getTriggerStreamsList ()
    {
        return triggerStreamsList;
    }

    /**
     * Sets the trigger streams list to the specified value.
     */
    public void setTriggerStreamsList (Vector v)
    {
        triggerStreamsList = v;
    }

    /**
     * Returns the if condition of this Vertex.
     */
    public String getIfCondition ()
    {
        return ifCondition;
    }

    /**
     * Sets the if condition to the specified value.
     */
    public void setIfCondition (String s)
    {
        ifCondition = s;
    }

```

```

/**
 * Returns the output guard list of this Vertex.
 */
public String getOutputGuardList ()
{
    return outputGuardList;
}

/**
 * Sets the output guard list to the specified value.
 */
public void setOutputGuardList (String s)
{
    outputGuardList = s;
}

/**
 * Returns the exception guard list of this Vertex.
 */
public String getExceptionGuardList ()
{
    return exceptionGuardList;
}

/**
 * Sets the exception guards list to the specified value.
 */
public void setExceptionGuardList (String s)
{
    exceptionGuardList = s;
}

/**
 * Returns the exception list of this Vertex.
 */
public String getExceptionList ()
{
    return exceptionList;
}

/**
 * Sets the exception list to the specified value.
 */
public void setExceptionList (String s)
{
    exceptionList = s;
}

/**
 * Returns the timer op list of this Vertex.
 */
public String getTimerOpList ()
{
    return timerOpList;
}

/**
 * Sets the timer op list to the specified value.
 */
public void setTimerOpList (String s)
{
    //timerOpList = s;
    timerOpList = timerOpList.concat (" " + s);
}

/**
 * Returns the informal description of this Vertex.
 */
public String getInformalDesc ()
{
    return informalDesc;
}

/**
 * Sets the informal description to the specified value.
 */

```

```

public void setInformalDesc (String s)
{
    informalDesc = s;
}

/**
 * Returns the formal description of this Vertex.
 */
public String getFormalDesc ()
{
    return formalDesc;
}

/**
 * Sets the formal description to the specified value.
 */
public void setFormalDesc (String s)
{
    formalDesc = s;
}

/**
 * Returns the keywords of this Vertex.
 */
public Vector getKeywordList ()
{
    return keywordList;
}

/**
 * Sets the keywords to the specified value.
 */
public void setKeywordList (Vector v)
{
    keywordList = v;
}

/**
 * Returns the timers of this Vertex.
 */
public Vector getTimerList ()
{
    return timerList;
}

/**
 * Sets the timer list to the specified value.
 */
public void setTimerList (Vector v)
{
    timerList = v;
}

/**
 * Returns the informal graph description of this Vertex.
 */
public String getGraphDesc ()
{
    return graphDesc;
}

/**
 * Sets the graph description to the specified value.
 */
public void setGraphDesc (String s)
{
    graphDesc = s;
}

/**
 * Returns the generic list of this Vertex.
 */
public String getGenericList ()
{
    return genericList;
}

```

```

/**
 * Sets the generic list to the specified value.
 */
public void setGenericList (String s)
{
    genericList = s;
}

/**
 * Returns the spec requirements of this Vertex.
 */
public Vector getSpecReqmts ()
{
    return specReqmts;
}

/**
 * Returns intersection point of this vertex with the specified point.
 */
public Point getIntersectionPoint (Point p)
{
    if (isTerminator ())
        return getTerminatorIntersection (p);
    else
        return getOperatorIntersection (p);
}

/**
 * Returns the intersection point of this vertex with the specified point.
 * Called from getIntersectionPoint when this Vertex is a Terminator
 */
public Point getTerminatorIntersection (Point p)
{
    int x;
    int y;
    float slope;

    slope = (float) (p.y - getY ()) / (float) (p.x - getX ());
    if (Math.abs (slope) >= (1 / 1.4f))
    {
        if (p.y <= getY () - getHeight () / 2)
            y = getY () - getHeight () / 2;
        else
            y = getY () + getHeight () / 2;
        x = (int) ((float) (y - getY ()) / slope) + getX ();
    }
    else
    {
        if (p.x <= getX () - getWidth () / 2)
            x = getX () - getWidth () / 2;
        else
            x = getX () + getWidth () / 2;
        y = (int) ((float) (x - getX ()) * slope) + getY ();
    }
    return new Point (x, y);
}

/**
 * Returns the intersection point of this vertex with the specified point.
 * Called from getIntersectionPoint when this Vertex is an Operator.
 */
public Point getOperatorIntersection (Point p)
{
    // Distance from the point to the center
    double distance = p.distance (getX (), getY ());
    int x = getX () + (int) (((double) (getWidth () / 2) / distance)
        * (float) (p.x - getX ()));
    int y = getY () + (int) (((double) (getHeight () / 2) / distance)
        * (float) (p.y - getY ()));
    return new Point (x, y);
}

// 11/9/99 SYT
/**
 * modify for accept two type vertex name grammar
 * Creates the specification construct from its data structures.

```

```

* This part finishes the operator specification on the last part
* of the grammar file.
* @param hasId boolean value that specifies if this Vertex has a
* unique id.
* @return returns the string representation of the specification
* of this Vertex.
*/
public String getSpecification (boolean hasId )
{
    Enumeration en = null;
    String tmp;
    //delete this to fix " concate error " 11/21/99 SYT
    //String spec = "";

    //add flag to answer Is this type or operator? 11/28/99 SYT
    boolean isType = false;

    // add to correct "do not concate error" 11/21/99 SYT
    StringBuffer operatorSpec = new StringBuffer ();

    try
    {
        if (hasId)
        {
            if (GrammarCheck.isValid (getLabel(), GrammarCheck.ID))
            { // change this 11/21/99 SYT
                // spec.concat("OPERATOR " +getLabel () + "_" +getId () + "_"
                //+ (getIdExtension ()) + "\n");

                operatorSpec.append
                (" OPERATOR " +getLabel () + "_" +getId ()

                    + "\n");
                operatorSpec.append (" SPECIFICATION\n");
            }
            else
            // change this 11/27/99 SYT
            //spec = spec.concat ("OPERATOR " + getLabel () + "\n");
            {
                isType = true;

                String tempString=getLabel();
                int indexDot = tempString.indexOf(".");
                int indexLeftParentheses = tempString.indexOf("(") ;
                //change output format 5/1/00 SYT
                operatorSpec.append(" TYPE "

                    + tempString.substring(0,indexDot)

                    + "\n");
                operatorSpec.append (" SPECIFICATION\n");

                if( indexLeftParentheses == -1 )
                {
                    operatorSpec.append(" OPERATOR "

                        + tempString.substring(indexDot+1)

                        + "\n");
                }
                else
                {
                    operatorSpec.append(" OPERATOR "

                        + tempString.substring(indexDot+1,indexLeftParentheses)

                        + "\n");
                }

                operatorSpec.append(" SPECIFICATION\n ");
            }
        }

        if (genericList.length () != 0 )
            operatorSpec.append (genericList + "\n");
        if (((Vector) specReqmts.elementAt (0)).size () != 0)
            en = ((Vector) specReqmts.elementAt (0)).elements ();
        String input = "";
        for (Enumeration enum = inEdges.elements (); enum.hasMoreElements
            ());
        {
            Edge e = (Edge) enum.nextElement ();
            //if this is not a state edge 5/16/00 SYT

```

```

//if (!e.isStateStream())
//{
    if ( input.lastIndexOf ( " " + e.getLabel () + " " ) == -1 )
    {
        input = input.concat ( "          INPUT "
                               + e.getLabel () + " : " + e.getStreamType () +
                               "\n");
        //change 7/16/00 SYT
        if(en != null)
        {
            if(en.hasMoreElements())
            {
                //if ((tmp = (String) en.nextElement ().length () !=
                // 0)
                tmp = (String) en.nextElement ();
                if(tmp.length() != 0)
                input = input.concat ( "          REQUIRED BY " + tmp +
                                       "\n");
            }
        }
    }
}
//}

operatorSpec.append (input);
if (((Vector) specReqmts.elementAt (1)).size () != 0)
    en = ((Vector) specReqmts.elementAt (1)).elements ();
String output = "";
for (Enumeration enum = outEdges.elements ();
enum.hasMoreElements ());
{
    Edge e = (Edge) enum.nextElement ();
    //if this is not a state edge 5/16/00 SYT (7/16/00 corrected)
    //if (!e.isStateStream())
    //{
        if ( input.lastIndexOf ( " " + e.getLabel () + " " ) == -1 )
        {
            if (output.lastIndexOf ( " " + e.getLabel () + " " ) == -1)
            {
                output = output.concat ( "          OUTPUT " + e.getLabel ()
                                         + " : "
                                         + e.getStreamType () +
                                         "\n");
                //change 7/16/00 SYT
                if(en != null)
                {
                    if(en.hasMoreElements())
                    {
                        //if ((tmp = (String) en.nextElement ().length () !=
                        // 0)
                        tmp = (String) en.nextElement ();
                        if(tmp.length() != 0)
                        output = output.concat ( "          REQUIRED BY " + tmp
                                                + "\n");
                    }
                }
            }
        }
    }
}
//}

operatorSpec.append (output);
if (((Vector) specReqmts.elementAt (2)).size () != 0)
    en = ((Vector) specReqmts.elementAt (2)).elements ();

String state = "";
for (Enumeration enum = children (); enum.hasMoreElements ());
{
    DataFlowComponent d = (DataFlowComponent) enum.nextElement ();
    //if this is a state stream 5/16/00 SYT
    if (d instanceof Edge && ((Edge) d).isStateStream ())
    {
        //no repeated edge 5/16/00 SYT
        if (state.lastIndexOf ( " " + ((Edge) d).getLabel () + " " ) ==
            -1)
        {
            state = state.concat ( "          STATES " + ((Edge) d).getLabel
                                   ()
                                   + " : "

```

```

        + ((Edge) d).getStreamType () + " INITIALLY "
        + ((Edge) d).getInitialValue () + "\n");
//change 7/16/00 SYT
if(en != null)
{
    if(en.hasMoreElements())
    {
        //if ((tmp = (String) en.nextElement ().length () !=
        // 0)
        tmp = (String) en.nextElement ();
        if(tmp.length() != 0)
            state = state.concat ("          REQUIRED BY " + tmp +
                                "\n");
    }
}
}
}
operatorSpec.append (state);

if (exceptionList.length () != 0)
    operatorSpec.append (extractString (exceptionList, false));
if (met != null) {
    operatorSpec.append ("          MAXIMUM EXECUTION TIME " +
                        met.toString ()
                        + "\n");
    if (!metReqmts.isEmpty ())
        operatorSpec.append ("          REQUIRED BY "
                            + extractList (metReqmts) + "\n");
}
if (keywordList.size () > 0)
    operatorSpec.append ("          KEYWORDS "
                        + extractList (keywordList) + "\n");
if (informalDesc.length () != 0)
    operatorSpec.append (extractString (informalDesc, false));
if (formalDesc.length () != 0)
    operatorSpec.append (extractString (formalDesc, false));

// need judge type. 11/28/99 SYT
// operatorSpec.append ("    END\n");
if(isType)
{
    operatorSpec.append ("          END\n");
    operatorSpec.append ("          END\n");
}
else
{
    //change output format 5/1/00 SYT
    operatorSpec.append ("          END\n");
}

}
catch (Exception ex)
{
    System.out.println (operatorSpec + "\n" + ex);
}
return operatorSpec.toString();
}
//SYT
/**
 * Called from getSpecification.
 * Extracts the string parameter and reformats it
 * to add to the specification.
 */
public String extractString (String str, boolean moreSpaces)
{
    BufferedReader reader = new BufferedReader (new StringReader
        (str));
    str = "";
    String line = "";
    try
    {
        {
            while ((line = reader.readLine ()) != null)
            {

```

```

        // from exceptionGuardList, outputGuardList and timerOpList
        if (moreSpaces)
            str = str.concat ("          " + line + "\n");
        // from exceptions formal and informal description
        else
            str = str.concat ("          " + line + "\n");
    }
}
catch (IOException ex)
{
    System.out.println (ex);
}
return str;
}

/**
 * Extracts an idList which is represented as a Vector and
 * returns a String representation
 * of the idList so that it will have the form "id1, id2, id3..."
 */
public String extractList (Vector v)
{
    String str = "";
    Enumeration enum;
    if (v != null)
    {
        enum = v.elements ();
        if (enum.hasMoreElements ())
            str = new String ((String) enum.nextElement ());
        while (enum.hasMoreElements ())
        {
            str = str.concat (" , ").concat ((String) enum.nextElement ());
        }
    }
    return str;
}

/**
 * Deletes this Vertex.
 * Deletes all the children of this Vertex and also
 * deletes all the in and out Edges.
 */
public void delete ()
{
    //use DefaultMutableTree method children
    //to delete all children. SYT 12/30/99
    for (Enumeration enum = children (); enum.hasMoreElements ());
    {
        DataFlowComponent dfc = (DataFlowComponent) enum.nextElement (
        );
        //recursive to delete all. SYT 12/30/99
        if (dfc instanceof Vertex)
            ((Vertex) dfc).delete ();
    }
    for (Enumeration enum = inEdges.elements (); enum.hasMoreElements ());
    {
        ((Edge) enum.nextElement ()).delete (true);
    }
    for (Enumeration enum = outEdges.elements (); enum.hasMoreElements ());
    {
        ((Edge) enum.nextElement ()).delete (false);
    }
    inEdges.removeAllElements ();
    outEdges.removeAllElements ();
    removeFromParent ();
}

// 11/28/99 SYT
/** this is no longer used.
 * and has been replaced by getID()-1.
 * before delete this must related item on psdlBuilder.jj first.
 */
public int getIdExtension ()
{
    return idExtension;
}

```



```
// 11/28/99 SYT
/**
 * this is no longer used.
 * and has been replaced by getID()-1.
 * before delete this must releated item on psdlBuilder.jj first.
 */
public void setIdExtension (int num)
{
    idExtension = num;
}
} // End of the class Vertex
```

THIS PAGE IS INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] Luqi, V. Berzins and R. Yeah, "A Prototyping Language for Real Time Software", *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, October 1988, pp. 1409-1423.
- [2] Luqi and M. Ketabchi, "A Computer Aided Prototyping System", *IEEE Software*, March 1988, pp. 66-72.
- [3] Luqi, " System Engineering and Computer-Aided Prototyping", *Journal of Systems Integration, Special Issue on Computer Aided Prototyping*, Vol. 6, 1996, pp. 15-17.
- [4] I. Duranlioglu, *Implementation of a Portable PSDL Editor for Heterogeneous Systems Integrator*, Master's thesis, Naval Postgraduate School, Monterey CA., March 1999.
- [5] G. Kreeger, *Requirements Analysis and Design of a Distributed Architecture for the Computer Aided Prototyping System*, Master's thesis, Naval Postgraduate School, Monterey CA., September 1999.
- [6] F. Brooks, *Mythical Man-Month(Anniversary Edition)*, Addison-Wesley, 1998.
- [7] JavaTM 2 SDK, Standard Edition Documentation, <http://java.sun.com/products/jdk/1.2/docs/index.html>, September 2000.

THIS PAGE IS INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, Virginia 22060-6218
2. Dudley Knox Library.....2
Naval Postgraduate School
411 Dyer Road
Monterey, California 93943-5101
3. Chairman, Code CS.....1
Naval Postgraduate School
Monterey, California 93943-5118
4. Professor Man-Tak Shing, CS/Sh.....1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5118
5. Professor Thomas Wu, CS/Wq.....1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5118
6. Professor Luqi, CS/Lq.....1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5118
7. Professor Valdis Berzins, CS/Be.....1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5118

8. CAPT. Shen-Yi Tao.....6
- 12-5 Fengle Rd.
- Taichung City, 406
- Taiwan, R.O.C.